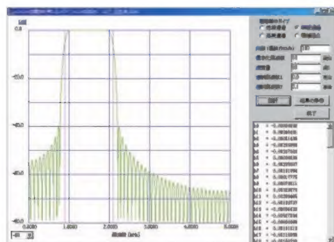
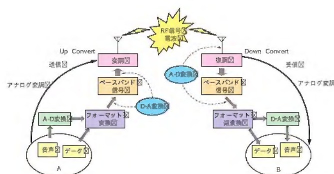
Interface

2004
9
Contents



［表紙デザイン：㈱プランニング・ロケッツ］





Interface

2004
9
Contents

話題のテクノロジ解説

ショウレポート&コラム

一般解説&連載

情報のページ

▶インターネット・セキュリティ・カンファレンス

RSA Conference 2004 Japan

北村 俊之

「ユビキタス時代を支える情報セキュリティ」をキーワードに，世界最大規模のデータ・セキュリティと暗号のカンファレンスである「RSA Conference 2004 Japan」が5月31日(月)～6月1日(火)の2日間，赤坂プリンスホテルで開催された(写真1)．主催はRSA Conference 2004 Japan 実行委員会．本カンファレンスは，2002年の第1回から数えて3回目の開催となる．



写真1　会場のようす

現在のディジタル社会，ネットワーク社会の急速な発展と広がりを考えると，セキュリティ分野での社会的な備えは十分とはいい難い．日本は現在，ブロードバンド大国といわれており，地上ディジタル波放送やモバイル通信をはじめとするユビキタス・ネットワーキングでも世界の先端を走っているといわれている．今後，日本が世界の情報セキュリティの実験場になる可能性があるという見方もある．

本カンファレンスでは，こうした最先端の情報分野の課題に対応することを重視するとともに，情報セキュリティの問題を企業や行政の経営管理の視点から捉えるために，新たにマネジメント・トラックが新設された．このマネジメント・トラックに加えて，ニュープロダクト＆テクノロジートラック，クラストラックなど，併せて60近くの講座が開催されていた．また，「サイバーセキュリティに関する官民の役割」と題した，米国土安全保障省全米サイバーセキュリティ局(NCSD)局長アミット・ヨーラン氏による基調講演をはじめとして，2日間で六つの基調講演が行われた．また，これらの各分野に関連した企業45社以上が参加した展示会も併催されるなど，セッション，展示会ともに年々スケールアップがはかられている．

● 展示会のようす

RSAセキュリティ(写真2)は，ITセキュリティのトップ・ベンダとして，ユーザ認証の「RSA SecureID for Microsoft Windows」をはじめ，Webアクセス管理の「RSA ClearTrust」，ディジタル証明書管理の「RSA Keon」，暗号化ツール・キット「RSA BSAFE」などの製品に関して，デモとプレゼンテーションを交えて紹介していた．参考出品の「RSA SecureID for Microsoft Windows」は，Windowsログオン時のユーザ名とパスワードの代わりにユーザ名と「RSA SecureID」が生成したパスコードを入力することで，オンライン環境，オフライン環境ともに強固な本人認証を実現するという．パスコードは，ユーザが記憶している4桁の暗証番号と「RSA SecureID」が生成する6桁の数字，合計10桁の数字で構成される．

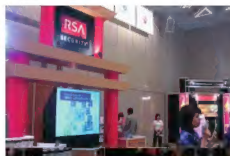


写真2　RSAのブース

ネットマークスは，セキュアITインフラ・システムをベースに，ワーム対策を中心とした検疫ネットワーク・システム，クライアントPC管理やログ管理などの情報漏洩防止対策，強固な認証を必要とするリモートVPNソリューションなどをデモを交えて紹介していた．また，顧客のネットワーク・システムをサポートするアウトソーシング・サービスとして「不正PC検出サービス」(写真3)の紹介を行っており，来場者の関心を集めていた．


写真3　ネットマークスの不正PC検出サービス

シマンテックのファイア・ウォール・アプライアンス「Symantec Gateway Security 5400 Series」(写真4)は，アプリケーション・レベルの攻撃に対応可能なフルインスペクション・ファイア・ウォール，プロトコル異常検知，シグネチャ・ベースに対応したハイブリッド侵入検知/防止システム，Gビットに対応した高パフォーマンスの実現，ポリシの自動配布，イベントの集中管理を実現するスケーラブルな中央管理などの特徴をもち，来場者の注目度も高いという．


写真4　シマンテックのファイア・ウォール・アプライアンス

マクニカネットワークスは，SSL-VPNの中でも評価の高い「Juniper NetScreenSA シリーズ」(写真5)や「バラクーダ スパムファイアウォール アプライアンス」の展示を行っていた．「バラクーダ スパムファイアウォール アプライアンス」は，10階層のフィルタリング技術でスパム・メールに対してブロック，隔離，タグ付け処理を行うことができる．最上位モデルでは，25,000のアクティブ電子メール・ユーザと，2,500万通/日のメッセージを処理することが可能だという．ソフトウェア・ソリューションと異なり，スパムの負荷をメール・サーバの前段階で取り除くことができ，サーバへの負荷を大幅に軽減できることが大きな特徴とのことであった．


写真5　マクニカネットワークスのJuniper NetScreenSA

オープンループは，データベース情報の漏洩による損害を防ぐソフトウェア・ソリューション「AdminSafe」の展示デモを行っていた．同ソフトウェアは，データベースの項目の内，個人の特定が可能な項目のみを自動的に暗号化することで，最小限のシステム負荷で個人情報漏洩による損害を防ぐことを可能にしている．既存のシステムに若干の手直しを行うだけで導入が可能，内部で自動的に暗号化/復号化を行うため，暗号化を意識する必要がないことなどを特徴としている．

日本ルーセント・テクノロジーズは，「Lucent VPN Firewall Brick ファミリ」(写真6)によるネットワーク・セキュリティ，および「NavisRadius 認証サーバ」による無線LANセキュリティの各ソリューションを展示していた．「Lucent VPN Firewall Brick ファミリ」は，LSMSによる一元管理型のVPNファイア・ウォール製品であり，最大1,000台のBrick，および最大10,000のLucent IPSec Client接続からのリモート接続を可能にしている．DDoS攻撃に対するプロテクション機能として，SYN floodプロテクション機能，インテリジェント・キャッシュ・マネジメント，TCPステート検証，フラグメントの再組み立てなどを搭載している．また，冗長構成機能を標準でサポートしており，400msのフェイル・オーバ時間を実現している．


写真6　日本ルーセント・テクノロジーズのLucent VPN Firewall Brickファミリ

ネットワーク関連の展示会

SUPERCOMM 2004 Chicago

松本 信幸

今年からはシカゴで

「SUPERCOMM 2004 Chicago」は6月22日～24日まで，今年から場所が変わり，シカゴで開催された(昨年までSUPERCOMMはアトランタ)．このため正確な雰囲気の差はわからないが，漠然と昨年より活気を取り戻してきているように感じた．実際，出展社数(675社)は昨年より2割増加しているとのことである．

しかし，通信業界としては依然厳しい状況が続いていることからか，目新しいソリューションを見ることはできなかった．そのため，とりあえず，先月にラスベガスで開催されたNetworld＋Interopで興味を引いたキーワードから比較して見てみることにする．



写真1　D-linkの監視クライアント

ワイヤレスLAN

アクセス・ポイントや，従来とおなじ系列のクライアント機器については，目を引くものはなかったが，カメラを無線端末として使用するというものがあった(写真1)．これはカメラにIEEE802.11bのアンテナをつけ，アクセス・ポイント配下において遠隔監視を行うというものである．

無線関係でもう一つ目を引いたものは，IEEE802.16(WiMAX)の機器が出始め，デモも行われているというものだった(写真2)．



写真2　RedlineのIEEE802.16関連機器

VoIP

VoIPについては，低価格化の流れのほかに，テレビ・カメラ付きのテレビ電話も出始めていた(写真3)．また，端末全体を見渡してみると，呼制御のプロトコルがSIPばかりではなく，MGCPやH.323をいまだに見かける．さらにVoIPのキーワードは端末ばかりではなく「Class 4/5 Switch」という，いわゆる局用に位置する装置についても多く出展されていた．

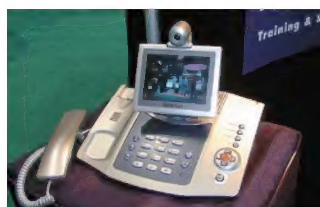


写真3　テレビ・カメラ付きVoIP対応テレビ電話端末

VoIPに関してもう一つ気になったのは，CODECとしてITU-T G.722のような広帯域(今までの上限3.4kHzではなく7kHz位までを範囲としている)をサポートしているものや，カンファレンス・コール(三者通話)が可能なものもそれなりに出て来ていたということである(写真4)．


写真4　Grandstream NetworksのIP電話

セキュリティ

展示会の性格が，コンシューマ向けではなく，どちらかといえばキャリア向けであるため，セキュリティ製品はあまり出ていなかった．前述のWLANカメラなども，考え方によってはセキュリティに入れても良いかもしれないが，少なくともセキュリティをキーワードに標榜した新しいソリューションや機器はほとんど目にしなかった．ただし，強引にいえばセキュリティ関連機器に入れても良さそうなものはいくつか出展されていた．もっとも興味を引いたものが「セッション・ボーダ・コントローラ」である．これは，VoIPサービスを提供するキャリアのネットワークについて，インターネットとの間(つまりファイア・ウォールが配置される部分)に位置され，キャリア相互間の接続時には，必要に応じてプロトコルの変換を行い，外部からの攻撃についてはファイア・ウォールの役割を果たすというものである(写真5)．


写真5　NETRAKEのセッション・ボーダ・コントローラ

そのほかの展示

展示内容を全体的に見渡して感じたことは，昔見かけたキーワードについて再度見るようになったものがあるということがある．これは技術が進展していないのではなく，昔登場した技術が最近になってようやく脚光を浴び始めたということであろう．表立ったキーワードではなく，よく見ないと見つけられないようなものではあるが，たとえばATMという単語がそこここに登場していた．ATMをバックプレーンとした高速ルータが新製品として登場してくるといった，5年くらい前にも見かけたような感じのする製品群が見うけられたのである(写真6)．無論，性能が5年前と同じということはない．

ATMが再登場した背景には，VoIPソリューションの本格化が大きく影響していると思われる．アプリケーションを提供するために必要なサービスの一つにQoSがあるためである．最初は「ATM並のQoS」と比較のためにことばを用いているのかと思えば，ファブリック・スイッチには実際にATMを使っていると説明を受けた．



写真6　Net.Comの高速ルータ

昔，ATM交換機の開発を行った者として，ATMという技術が今でも生きていることはうれしいものではあるが，その中に日本企業がほとんど入っていないのは寂しくもある．

もう一つ目を引いたものとして，電源(特にバッテリ)関連があった．その中でもPoE(Power over Ethernet)向けに，ラック搭載型のバックアップ用バッテリがあり，商用電源からの電力をいったんバッテリにチャージしながら供給し，停電の際にもPoE経由でIP電話機の電力を供給できるようにしようというものであった(最終日に写真を撮りに行ったが，残念ながら撤収した後で撮影はできなかった)．

いまだ迷走している感は否めないとはいえ，再び活気が出てきていることは喜ばしく，来年も期待できそうだった．

まつもと・のぶゆき　(株)タムラ製作所

Show & News Digest

TOPPERSカンファレンス2004

■日時：2004年6月3日（木）～4日（金）
■場所：学士会館（東京都千代田区）

NPO法人TOPPERSプロジェクトにより，フリーなμITRON実装であるTOPPERSに関するカンファレンスが開催され，200名近い参加者を集めた．

TOPPERSプロジェクト会長の高田 広章氏による「TOPPERSプロジェクトの最新状況」では，これまで開発してきたTOPPERS/JSPが1.4で完成の領域に達したと判断し，JSPはここでフリーズしたいとのことだった．これにより次のバージョンである1.5はリリースせず，フルセット・カーネルであるTOPPERS/FI4に注力したいというロードマップを示した．なお，JSPも新CPUへの対応版である1.4.1などのマイナ・バージョンアップは行いたいという．また，NPO法人として発足してから1年が経過したが，初年度に約10点の開発成果を配布開始/発表できたことについて，会員の協力に感謝するとともに，NPO法人としては十二分な開発成果が得られたのではないかとも語った．

また，現在進行中のプロジェクトとして，マルチプロセッサ対応TOPPERSをあげ，完成度は高くはないものの，すでに動く実装ができているとの状況を明らかにした．さらにプロセッサ時間の保護機能など，組み込み機器では有効に使える機能などにも取り組みたいとのことだった．



TOPPERSプロジェクト会長の高田 広章氏

ウインドリバー，Linux対応の統合開発環境Wind River Workbench 2.0を発売

■日時：2004年6月22日（火）
■場所：ウインドリバー本社（東京都渋谷区）

ウインドリバー（株）は，組み込みLinuxに対応した統合開発環境Wind River Workbench 2.0を発表し，出荷を開始した．これまでVxWorksをあつかっていた同社がLinux製品を発表するということについて，代表取締役社長の藤吉 実知和氏は，「Wind Riverは顧客にソリューションを提供するという立場から，VxWorksとLinuxの両プラットホームで統合開発が行えるWind River Workbenchは，戦略的な位置づけにある製品だ」と語った．また，同社がLinuxをあつかったことにより商談が増え，間接的にVxWorksの売り上げも伸びるといった現象も起きているとのことだ．これは最初はLinuxで製品開発を予定していた顧客が，製品仕様の絞り込みによりVxWorksのほうが適切であると判断して，使用するOSを変更するというケースが多いためだとのことだ．

Wind River Workbench 2.0は統合開発環境Eclipseへのプラグインの形で提供される組み込み向け開発環境．Linuxホスト上で動作し，ターゲットとしてはPowerPC MPC82xxシリーズ上で動くLinuxカーネル2.4.22ほかを対象としている．通常のLinuxをターゲットとしたデバッグは，カーネルを対象とする場合にはkgdbサーバ，アプリケーションを対象とする場合にはgdbサーバを使う必要があり，それぞれを切り替えてデバッグする必要があった．Wind River Workbenchに含まれるWDBデバッグ・エージェントは，この二つを統合し，カーネルのデバッグとアプリケーションのデバッグをシームレスに行えるようにしたものだ．会場ではブレーク・ポイントを設定して任意の位置で実行を停止するデモが行われ，カーネル内でもアプリケーション内でも，同様にブレーク・ポイントを設定できることをアピールしていた．なお，このWDBデバッグ・エージェントは，Linuxカーネルをビルドするときにあらかじめ組み込んでおくことでリモートからのデバッグを可能にするスタブである．

今回の発表の時点では，ホストとしてLinuxのみに対応しているが，9月にWindows，年内にはSolaris対応版も出荷予定とのこと．また，ターゲットOSとしてVxWorksにも年内に対応する予定で，VxWorksでもWind River Workbench上で同様のデバッグ環境を得られることになる．ターゲットCPUに関しても対応を順次拡大予定という．

代表取締役社長 藤吉 実知和氏



デバッグ環境の例

アルテラ，低コストFPGA Cyclone II を発表

■日時：2004年6月28日（月）
■場所：新宿センチュリーハイアット東京（東京都新宿区）

日本アルテラ（株）は，低コストFPGAファミリCyclone IIを発表した．90nm低誘電率材（Low-K）プロセスで製造され，従来のCycloneと比較して3倍以上のロジックと4倍のメモリをもち，最大150個の18×18乗算器，DDR2/QDR IIインターフェースを提供したうえで30%の低価格化を実現した．

ロジック・エレメント数は4,608～68,416個で，50,528ロジック・エレメントのデバイスEP2C50の価格は$50を予定している．



Cyclone II

ICカード付き携帯電話が作る新しい文化

山本 強

今度の携帯電話には，FeliCa仕様の非接触ICカード機能が組み込まれるという発表があった．携帯電話を使った電子決済は，"フィンランドでは携帯電話でコーラが買える"という話に始まって，国内外であらゆるモデルが試されてきているのだが，これまでのところ，ビジネスとして成立するシステムは出ていなかったように思う．今回発表されたFeliCa携帯電話は，少なくとも現時点で電子マネーとして流通している決済システムと互換性があり，使える場所がすでにあるという点でも画期的なものだといえる．

携帯電話が電子マネー機能を持ったと聞くと，何か画期的な技術が開発されたかのように聞こえる．しかし冷静に分析すれば，既存のICカードを携帯電話に貼り付けただけという見方もできるわけで，IT技術者の視点では技術的なインパクトはあまり感じられない．しかし，携帯電話がICカードだけでなくリーダ/ライタとしての機能をもっていることを考えると，これは携帯電話やICカードの「文化」という点で画期的ではないかと思うのである．

基本は反応速度と使用中の形

携帯電話を決済システムにする話はいろいろ考えられているのだが，今ひとつ普及していない．その理由の一つに，「人を含めたトランザクション速度」という問題がある．2次元バーコードは端末側に追加コストがほとんどないという利点があるのだが，画面にコードを表示して，それを「上手に」読み取る機械にかざすという操作に10秒以上かかるという問題がある．表示されたコードはリーダが読み取っても画面に残るから，必然的にクローズ・ループのサービスとなり，サーバへのアクセス時間も無視できない．携帯電話決済で現金を置き換えることを考えるなら，小額の現金払いに要する時間，つまり，お金を出す→数える→おつりを出すという一連の動作に相当するトランザクションを，10秒以内で完結できなければ受け入れられないだろう．この点，ICカードはすでにJRなどの改札に使えるくらいの応答速度を達成している．

もう一つ，「形」という問題がある．これは携帯電話の静的な外形デザインではなく，使用中の「人の形」である．これまで携帯電話にいろいろな機能が標準装備されてきたが，それを使っている人をあまり見ないものがある．たとえば，音声認識電話番号検索である．認識精度という問題もあるのかもしれないが，この機能をオフィスで使っている人の見え方には，かなり違和感があるし，一人のときに電話に向かって一定の調子で番号や人の名前を発声するのはとても怪しい．どうしても手が使えないといった特殊な状況での需要しか出ないのではないかと思うのである．ICカードは反応時間が短いのが特徴なのだが，これは一瞬で終わるから多少変な動作でも目立たないということにつながるのである．

接触という動作が作る新しい形

技術的に見れば，ICカードとリーダ/ライタが一個体にパッケージ化されているわけだから，理屈では携帯どうしで接触している相手のICカード情報を読み書きできることになる．残念ながら発表されている限り，そういったサービスは提供されないようだが，それができるといろいろと新しい携帯文化ができてくる予感がする．

たとえば，電話番号の交換である．今は互いに「ワンコ」するのが定番であるが，これは二つの携帯を接触させるという動作に変えることができる．それで氏名，電話番号，メール・アドレスなど公開指定している情報が交換できるようになる．これを「携帯キッス」と命名する．CMやトレンディ・ドラマの中にこのシーンをさりげなく刷り込んでおくことで，案外簡単に定着させることができそうである．

番号交換の延長上に電子マネーや電子チケットの個人交換ということも出てくる．SuicaやEdyが提供する電子マネーはオープン・ループ型であり，キャッシュに相当するデータはICカード間で転送することが可能である．だとすれば，携帯電話の接触により，価値を移すということもできるということになる．これは通貨という国家の基幹システムに挑戦するという，危ない領域に踏み込む話ではあるのだが，冷静に考えればすでに金券屋というシステムで商品券だのチケットだの換金性のある擬似貨幣が普通に流通しているという現実もあるわけで，案外容易に実現できる話ではないかと思えてくる．

その先にあるP2Pの電子マネー

つまり，ICカード対応携帯電話のしくみは個人間の価値交換機能を潜在的にもっているのである．たとえていえば，P2Pの電子マネーである．現金が他の決済システムに対して圧倒的に勝っていたのがこのP2Pで匿名の決済なのだから，電子決済が現金の領域に入り込もうとするならば，個人間の価値の取り引きができるかが重要になってくる．ICカード対応携帯電話はシステムとしてその領域にあることはまちがいない．

P2Pのファイル交換が著作権侵害という社会問題を起こしたように，P2Pのマネー交換も社会問題を引き起こす可能性がある．不良学生が相手を脅して携帯電話の電子キャッシュを脅し取る，「携帯カツアゲ」がその先の社会現象とならないことを願いたい．

やまもと・つよし　北海道大学大学院情報科学研究科
メディアネットワーク専攻
情報メディア学講座

FIRフィルタ
1kHz矩形波

FIRフィルタ
デモ

Effect
ハイパス

今こそ基礎を身に付け，チカラを上げる

特集 原理から学ぶ ディジタル信号処理技術

Effect
8近傍
Laplacian

ディジタル信号処理は，通信や音声処理，画像処理，制御，計測など，幅広い分野で当たり前のように使われている技術であるため，今，この技術をしっかりと身に付けた技術者が求められています．そこで本特集では，ディジタル信号処理技術の基本を原理から徹底して解説します．

はじめに，今，この分野に何が求められているのかを，DSPの歴史を振り返りながら探ります．次に，ディジタル信号処理において重要な位置付けにあるディジタル・フィルタについて，そのアルゴリズムから解説します．さらに，音声処理と画像処理のアプリケーションの作成をとおして，1次元および2次元のフーリエ変換や，さまざまなディジタル・フィルタを取り上げます．ここで作成したソフトウェアやシミュレータは，本誌のWebサイトからダウンロードできるので，それを使って実際にディジタル・フィルタを体感してみてください．また，急速な発展を続けているディジタル無線通信についても取り上げ，そこで行われているデータの変調や復調方法などについても紹介します．

Prologue カラーで見る今月の特集

今月の特集は，ディジタル信号処理技術の原理の部分に焦点を当てている．本題に入る前に，まず，特集の内容を一通り紹介していく．

（編集部）

第1章 ディジタル信号処理技術の歴史
図1，図2，図3

第1章では，ディジタル信号処理専用のプロセッサである DSP（Digital Signal Processor）の歴史を振り返りながら，今，この分野に何が求められているのかを探っていく．

世界で最初に商用化された DSP は，日電（NEC）の μPD7720 で，1980年に発表された．

オーディオなどのアナログの分野がディジタル化されていく一方で，汎用のマイコン（Z80など）では，ディジタル・フィルタの実時間処理が困難であり，かといって高性能なハードウェアを使えばコストが跳ね上がる…という状況の中で生まれた．当時のエンジニアの多くは，DSP のアルゴリズムが数学的な色彩が強かったため，かなり戸惑ったそうだ．

それから約24年が経過した今，DSP はディジタル携帯機器の広まりで出荷数が爆発的に増加し，ハイエンドのものでは動作周波数が 1GHz を超えた．

また，開発環境についても，少し前まではアセンブリ言語を用いての開発が一般的だったが，最近では C++ 言語を用いて開発するケースも出てきた．

第2章 FIRフィルタの設計と実現方法
図4，図5

第2章では，ディジタル信号処理において重要なポジションにあるディジタル・フィルタについて，その原理から設計，DSP への実装までを解説する．なお，ここでは，FIRフィルタの設計のために筆者オリジナルのツールを使っている．このツールは本誌の Web サイトからダウンロードできる．

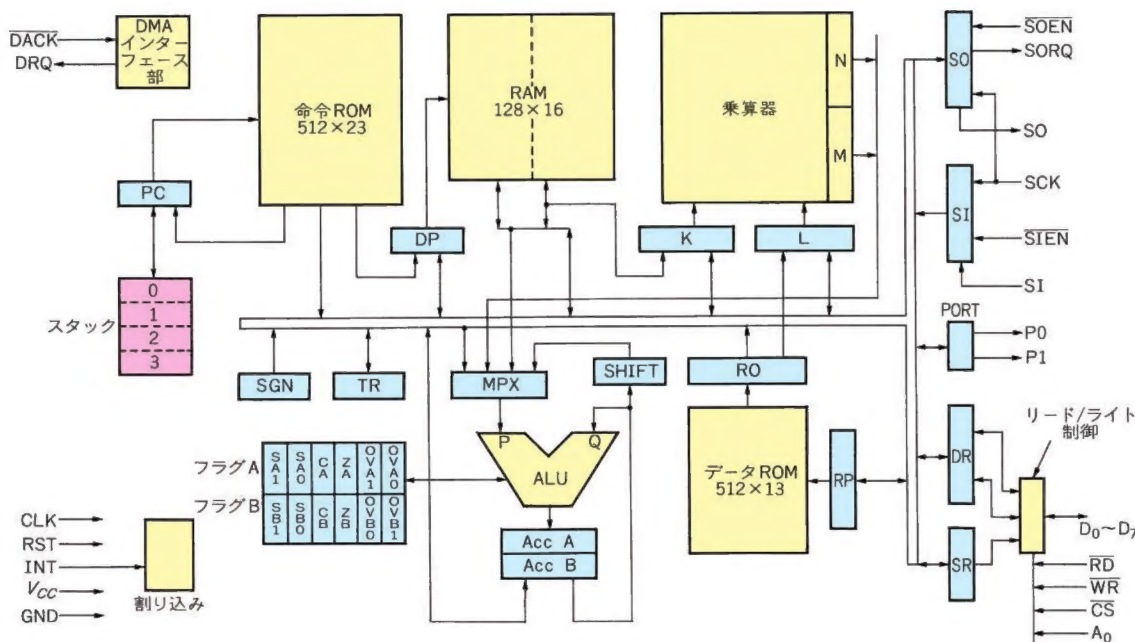


図1 μPD7720のブロック図



図2　動作周波数が1GHzを超えたTI社のDSP
（TI社のTMS320C64Xシリーズ）

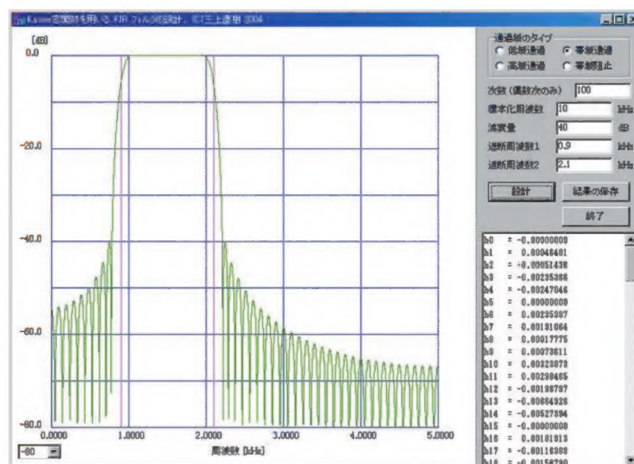


図4　筆者が開発したFIRフィルタの設計ツール

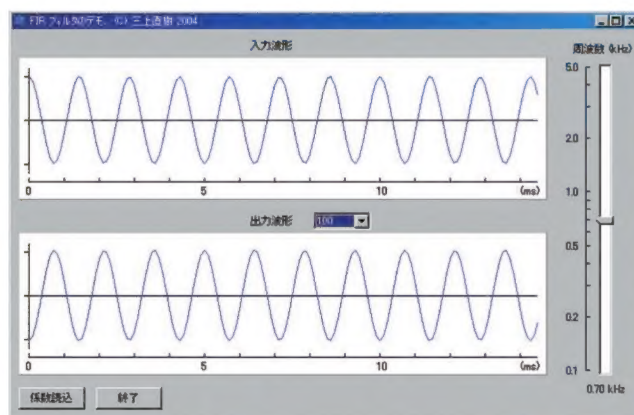


図5　筆者が開発したFIRフィルタのデモ用アプリケーション

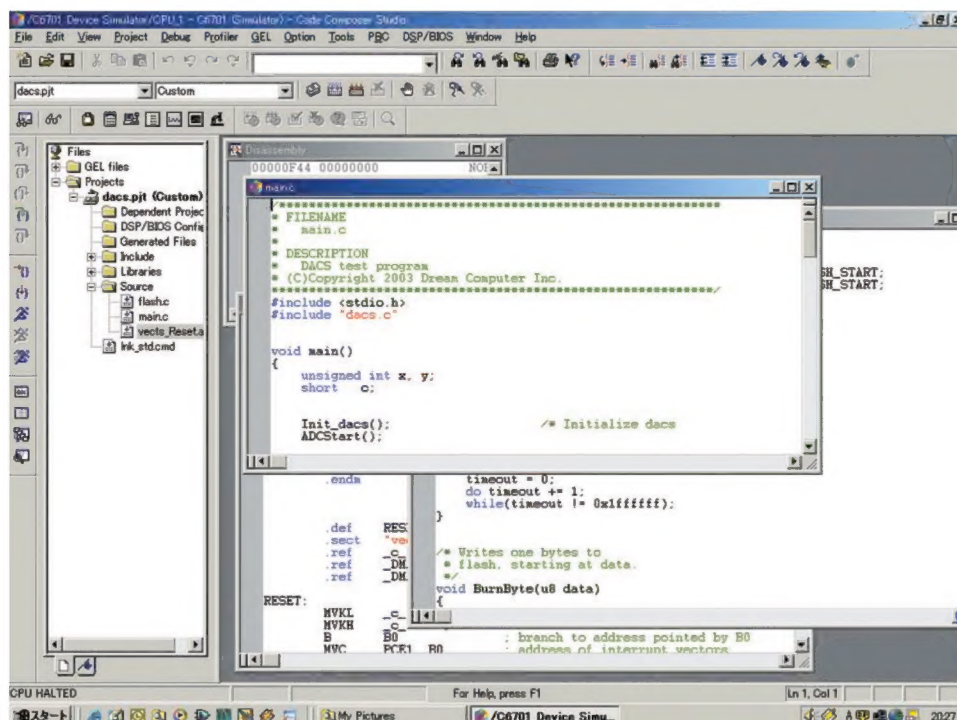


図3
C++言語を使用するDSPの開発環境
（TI社のCode Composer Studio）

Pro logue
1
2
3
4
5
Appen dix

New Products ——チャージ・ポンプ方式利用のTFTディスプレイのバックライト・ドライバ
アナログ・デバイセズ社は，携帯電話，PDA，ディジタル・カメラなどのカラーTFT液晶パネルに使用されるバックライト駆動を高精度で行う，バックライト・ドライバ「ADM8845」を発売した．価格は$0.70（1万個時）．

第３章　DSPで実現する音声処理アプリケーションの開発　図６

ディジタル・フィルタの原理の理解はもちろんだが，実際に体感してみようということで，カラオケなどでもおなじみのディレイ(エコー)やリバーブなどのエフェクタをDSPボードを用いて作成する．ここで作ったエフェクタのシミュレータが本誌のWebサイトからダウンロードできる．これを利用して，ぜひディジタル・フィルタを耳で聴いて感じてほしい．

第４章　画像処理アプリケーションの作成　図７，図８，図９

フーリエ変換からの展開で，２次元FFTを中心に解説する．そして，この２次元FFTを理論だけではなく，感覚的にも理解するために，簡単な図形認識のプログラムを作成した．このプログラムも本誌のWebサイトからダウンロードできるので，これを使って２次元FFTとはどのようなものなのかを体感してみてほしい．

第５章　ディジタル変調/復調の基礎と原理　図10

ここでは，アナログからディジタルへの移行という視点で，近年，発展の目覚しい無線通信分野で用いられるディジタル信号処理技術について解説する．また，この技術を実現するために用いられているハードウェアについても述べる．

それでは…　図11

ディジタル信号処理技術の原理を学び，基礎をしっかりと身に付けて，新しいアルゴリズムを考え出せるようになれば，それだけ開発できる製品の幅も広がっていく．本特集をきっかけに，そのようなワン・ランク上の技術者をめざしてほしい．

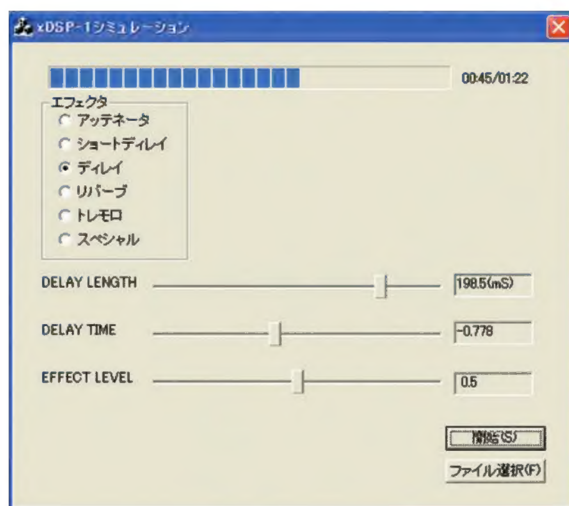


図６　エフェクタのシミュレータ

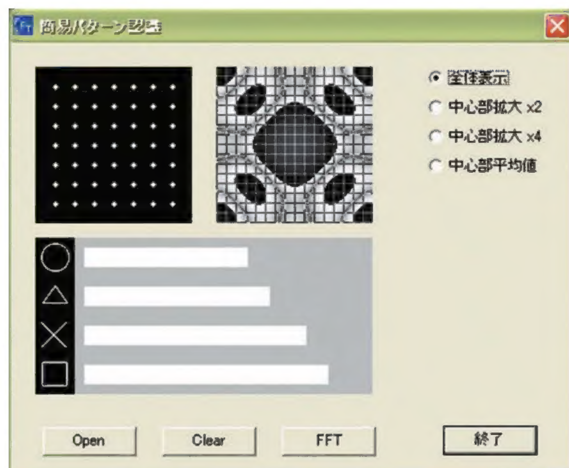


図７　作成した画像処理アプリケーション

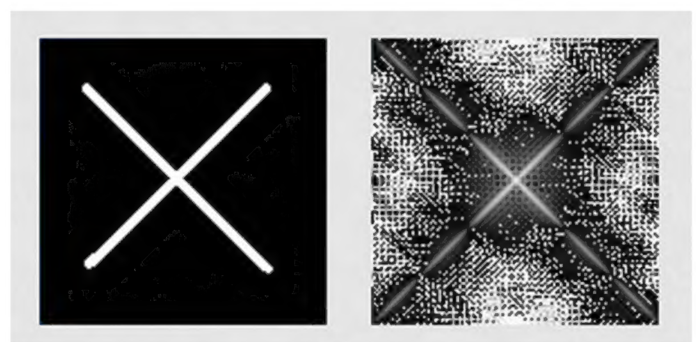


図８　２次元FFTの結果(1)
左が元の画像で，右がFFTによって得られたスペクトル

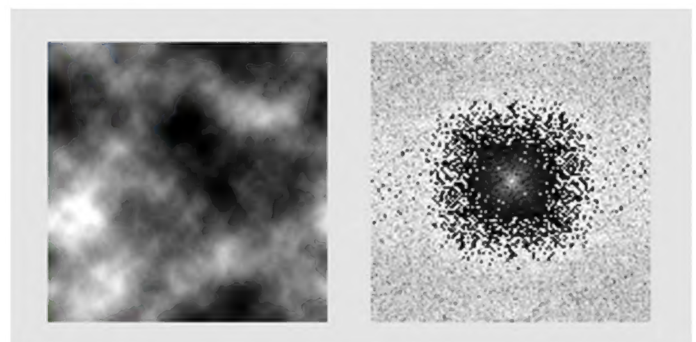


図９　２次元FFTの結果(2)
スペクトルの中心がDC成分で，両端に行くほど高い周波数領域を表している．そのほかの結果はp.86へ

New Products ――ミューチップを活用した業務アプリケーションの開発支援ソフトウェア
(株)日立製作所 情報・通信グループは，非接触ICチップ「ミューチップ」を使ったRFIDタグを活用し，個品の履歴管理や追跡照会などの各種業務アプリケーションの開発を支援するソフトウェア「ミューチッププマネージャ」を発売した．

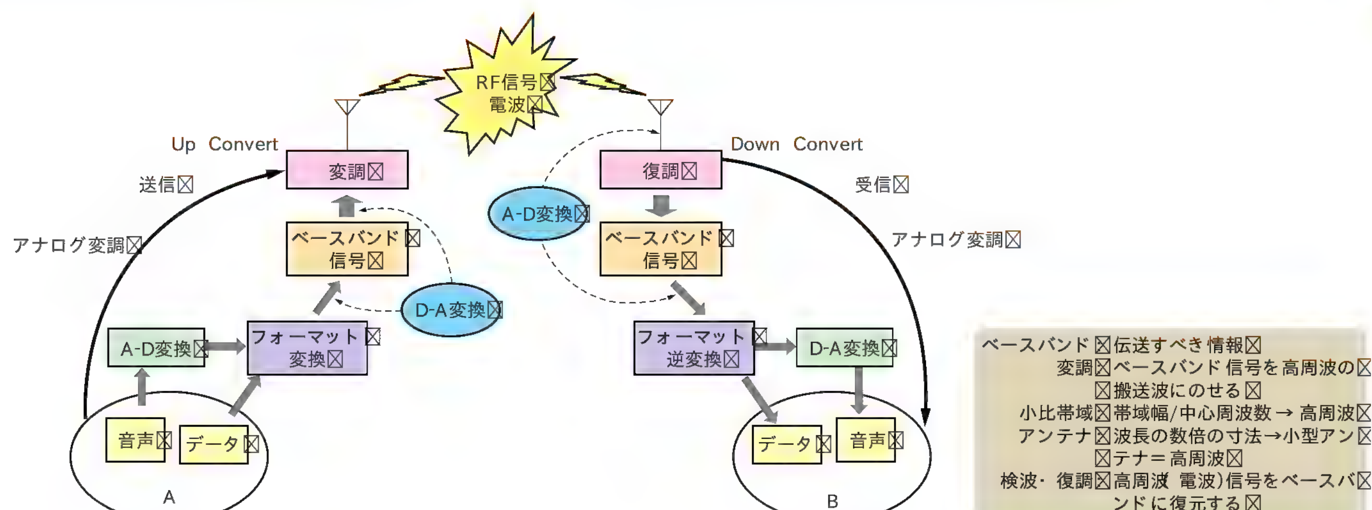


図10　ディジタル無線通信

Pro logue
1
2
3
4
5
Appendix

図11　第5章の筆者がディジタル信号処理技術を駆使して開発した「超掃引式スペクトラム・アナライザ」

「超掃引式スペクトラム・アナライザ」で観測したスペクトラム．SPAN=5MHz，RBW=1kHzを1s弱で測定（従来方式の10倍速）．背面がSPAN=5MHz．前面はその一部．SPAN=500kHzを部分拡大表示したもの．最新の信号処理デバイスと，アルゴリズムを駆使して開発したもの．信号処理の原理をしっかりと学んだからこそ，開発できた製品である

New Products――ナショセミ，超薄型ICパッケージを発表
ナショナル セミコンダクター ジャパン（株）は，厚さ0.4mmの超薄型ICパッケージを発表した．新パッケージは，micro SMD（表面実装デバイス）とLLP（リードレス・リードフレーム・パッケージ）をさらに薄型にしたもの．

1 DSPの変遷から見る

ディジタル信号処理技術の歴史

DSP(Digital Signal Processor)は，ディジタル信号処理の演算を高速に行うことに特化したLSIである．このDSPの変遷には，その時代ごとのディジタル信号処理技術への要求や歴史が映し出されている．

そこで，本特集のスタートとなるこの第1章では，DSPを通じてディジタル信号処理の歴史の流れを振り返ることで，今，この分野に何が求められているのかを探っていく．

(編集部)

大久 信広

1 初めてのリアルタイム信号処理

筆者の手元に「シグナルプロセッサとその応用ーTMS320」という本がある．これは，1986年にコロナ社より発行されたものである．この1986年当時，筆者は音声処理装置の開発に取り組んでおり，NEC社のPCであるPC-9801と日々向かい合っていた．

努力の甲斐あって，アコースティック・ツール・キットという波形解析ソフトウェアが完成した．これをツールにしてリアルタイム音声認識や音声合成アプリケーションの開発という，当時としてはとてつもなく無謀な計画に取り組むこととなった．というのも，PC-9801では性能に限界があったからである．

しばらくして，やはりPC-9801の速度不足やメモリ不足という壁にぶつかり，本格的なリアルタイム音声アプリケーションの開発が絶望的になりつつあった．そしてその矢先に先の本で「DSP」なるものの詳細を知ることとなったのである．このDSPとは「Digital Signal Processor」の略で，まさにディジタル信号処理のためだけに特化したプロセッサなのだ．

PC-9801ではなぜリアルタイム音声処理ができないのかというと，先ほども述べたとおり，ずばり速度不足ということにつきる．当時，すでにディジタル信号処理の理論に限っていえば現在とほぼ同等のレベルに達していた(1)．それにもかかわらず，理論を実行するためのハードウェアについては現在と比べようもなく貧しい環境しか手に入らなかったのである．現在では，ポケット・マネー程度(？)で購入できるDSPが，なんとクロック周波数が1GHz，演算処理速度については8000MIPSという高い性能にまでに達している注1．

さらに，それでも不足の場合は，大容量のFPGA(Field Programmable Gate Array)を用いることで，個人でも専用のハードウェアを備えたプロセッサを作成することもできるようになった．それもたかだかここ20年の間の出来事である．

この第1章では，ディジタル信号処理の歴史を理解するために，それぞれの時代に脚光を浴びたDSPを紹介し，その時代ごとのディジタル信号処理への要求の変遷を振り返ることとする．そして今，ディジタル信号処理に何が求められているのかを明らかにしていこう．

2 最初のDSP――第1世代DSP

Intel社の8080とMotorola社のMC6800がしのぎを削っていたころから少し遅れ，DSPの開発がスタートし，1980年には日電(NEC)から世界最初の商用DSPであるμPD7720(写真1)が登場した．また，それから2年後には米国Texas Instruments社(以下TI社)のTMS320C10(写真2)が発表された．これはそれまでのアーキテクチャ(図1)とは異なるハーバード・アーキテクチャ(図2)と呼ばれる，データ・バスとプログラム・バスを分離した，当時としては画期的なDSPで，以後のDSPの開発に大きな影響を与えることとなった．このころのDSPを第1世代DSPと呼ぶ(表1)．第1世代DSPは，音声圧縮処理やエコー・キャンセラ，モータ・コントローラといった汎用アプリケーションをターゲットにしたものだった．

これらのDSPが登場する以前は，たとえばFFT演算処理が必要な場合，数十万円もする乗算器とランダム・ロジックを組み合わせてFFT演算処理を行っていた．開発資金が潤沢な場合は，専用のチップを開発するというケースもあった．いずれにしても，ディジタル信号処理の実現にはたいへんな開発コストと時間を必要とする時代が続いていたのだった．

しかし，DSPの出現により状況は劇的に変化した．安価なDSPでソフトウェアによるディジタル信号処理が実現可能となったため，開発エンジニアは煩雑なハードウェア設計から解放された．それから開発コストが引き下げられたため，多くの

注1：米国Texas Instruments社のTMS320C6414T/15T/16T．

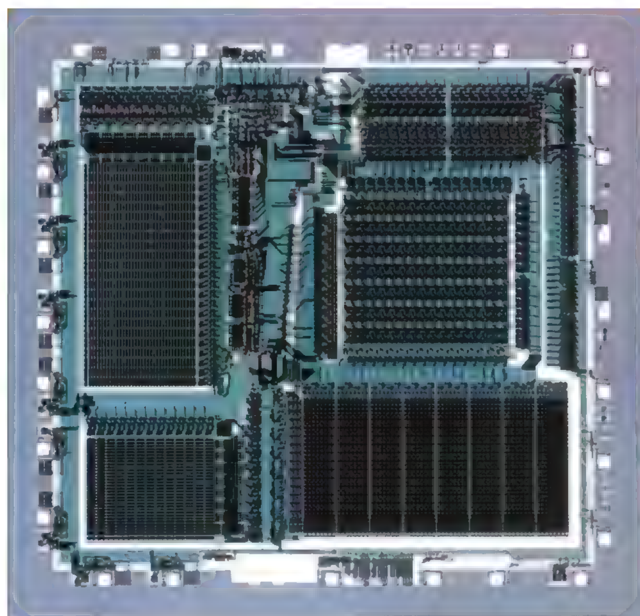


写真1　世界最初の商用DSPである「μPD7720」(NEC)の回路パターン

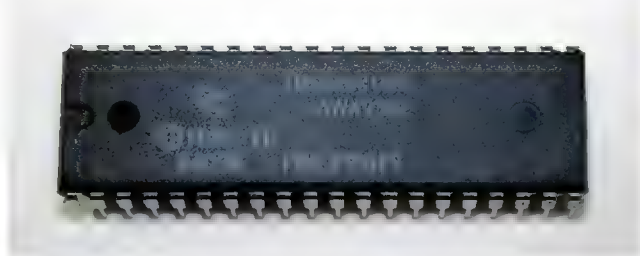


写真2　ハーバード・アーキテクチャを採用したTMS320C10(TI社)

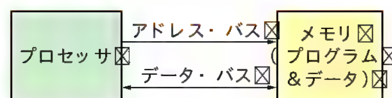


図1　それまで一般的だったアーキテクチャ

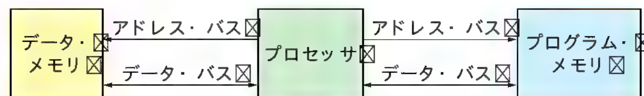


図2　ハーバード・アーキテクチャ

表1　代表的な第1世代DSP

年	開発元	型　番	備　考
1980	ベル研究所(現在のAT&T)	DSP20	世界最初のDSP．実用に至らず
1980	NEC	μPD7720	日本で最初のDSP．世界最初の商用DSP
1982	TI社	TMS32010	最初のハーバード・アーキテクチャ採用のDSP
1982	日立	HD61810	――――――――――――――
1982	富士通	MB8764	当時，世界最速のDSPだった

表2　DSPの応用分野

応用分野	処理内容
汎用DSP	ディジタル・フィルタ，相関，波形発生
音声	音声認識，音声合成，テキスト音声変換
画像	画像圧縮，パターン認識
制御	ディスク制御，サーボ制御，モータ制御
計測	スペクトル解析，パターン・マッチング，地震波解析
軍事	レーダ信号処理，ソナー信号処理，ミサイル誘導
通信	エコー・キャンセル，適応等価器，DTMF符号化，復号化
自動車	エンジン制御，振動解析，適応サスペンション制御
家電	ディジタル・オーディオ，合成音声応答装置，ゲーム
産業	ロボティックス，数値制御
医療	補聴器，超音波診断装置，MRイメージング

表3　代表的な第2世代～第3世代DSP

年	メーカ	型　番	備　考
1986	AT&T社	DSP32	32ビット浮動小数点
1986	NEC	μPD77230	32ビット浮動小数点
1988	TI社	TMS320C30	32ビット浮動小数点
1987	Motorola社	MC56000	24ビット固定小数点
1987	富士通	MB86232	32ビット浮動小数点

分野でDSPが採用されることとなったのである．

命令サイクルが100ns程度と，現在とは比較にもならない第1世代DSPだったが，1990年初頭よりプロセスのPID制御，ハードディスクのヘッド・ポジショニング，ロボットのモーション・コントロール，ACインダクション・モータ・ドライバ，ACサーボ・ドライバ，ブラシレス・モータの適応制御，自動車タイヤのアクティブ・ノイズ低減など，堰を切ったようにディジタル信号処理にDSPを応用するという試みがスタートすることとなった．

表2に応用分野の一覧を示す．これを見ると，まさにあらゆる分野にわたってDSPの使用が一気に浸透していったことがわかる．

3 広がるDSPの応用 ――第2世代～第3世代DSP

第1世代DSPに引き続いて第2世代～第3世代のDSP開発および発表が1980年代の後半までに行われた(表3)．第2世代～第3世代DSPはそれまでのDSPに対し，データ幅およびメモリ空間の拡張，アドレシング機能の強化，DMA機能の標準装備といった，基本仕様を拡充させるための改良と浮動小数点演算機能の追加といったことが行われた．

一方，それとは別に第2世代～第3世代DSPとして，オーディオ信号処理や画像処理といった特定アプリケーションに特化したDSPも開発されるようになった．

たとえば画像信号処理として，図3に示すようなビデオ信号をディジタル信号処理で行う場合を取り上げると，8ビットで

Information――アルテラ，シノプシスとHardCopyストラクチャードASICで提携
アルテラ社とシノプシス社は，シノプシス・プロフェッショナル・サービスによるアルテラHardCopyデザイン・センタのサポートを発表した．この提携は，市場で急速に増え続ける量産に適したHardCopyデバイスの需要に対応するものである．

14.31818MHzという非常に高い周波数でサンプリングが行われることとなる．また，1画面を512×512画素，フレーム周波数を30Hzとすると7.864320Mバイト/sという膨大なデータを処理する必要がある．

このように画像信号処理は，処理速度の厳しさや，扱うデータ量の多さが障害となって，これまでの汎用DSPではなかなか歯が立たなかった．

そこで登場したDSPが，特定アプリケーションに特化したASSP(Application Specific Standard Product：アプリケーション指向汎用LSI)と呼ばれるものである．1988年に画像処理用ASSPとして，日立からDSP-I(HD81831：図4)が発表された．このDSPは画像処理を効率よく行うために，以下のような特徴を備えていた．

- サイクル・タイムが50nsと速い
- マイクロプログラムとピコプログラムの2階層の命令で並列処理を行う
- 2次元アドレス演算ユニットで，2次元データの効率の良いアクセスが行える
- ビット演算ユニットで画素データを効率良く操作できる
- プログラムRAMのほかに，4ページのデータ・メモリを実装し，画像データの効率良いアクセスが行える

このDSPの出現によって，信号の圧縮やノイズ除去，エッジ抽出のような画像信号処理がソフトウェアで実現可能となり，本格的なASSP時代の幕開けとなった．

もう一つのASSPの例として，FFT(Fast Fourier Transform)演算を実行するためのDSPを取り上げる．ZORAN社のZR34161(図5)というDSPは，VSP(Vector Signal Processor)と呼ばれ，FFT演算を効率よく実行するための専用ハードウェアとして機能する．このDSPの特徴として，実数部と虚数部およびサイン・テーブルを収めるための専用RAMが用意されており，またFFT演算では必ず行われるビット・リバーシングというデータの並べ替えも，アドレス・ジェネレータでサポートされていることが挙げられる．その結果，4096点のFFTを19.64msという速さで実行することが可能となっている．これにより，FFT演算を必要とするレーダやソナー，振動解析などの分野で大いに役立つこととなった．

当時，安価になったPCにこれらのDSPを拡張基板として装備すれば，実用的なリアルタイム画像処理システムやスペクトル・アナライザが驚くべき低価格で実現することが可能となった．このころからPCと各種機能の拡張基板を組み合わせて，さまざまなシステムを低価格で構築することが一般的となっていった．

4 最新のDSPの動向

ここからは，最新のDSPの動向を見ていくことにしよう．

● ディジタル・オーディオ規格に対応 ——ポータブル・オーディオ用途

MP3などのディジタル・オーディオ・ファイルによるインターネット上の音楽配布は，従来のCDベースでの音楽配信形態に変化を与えた．2003年に開始された米国Apple Computer社によるインターネットを通じたディジタル音楽配信は，1曲あたり99セントで購入できる手軽さが受け，販売曲数は開始1週間で100万曲を突破した．その後，11か月で販売楽曲数が

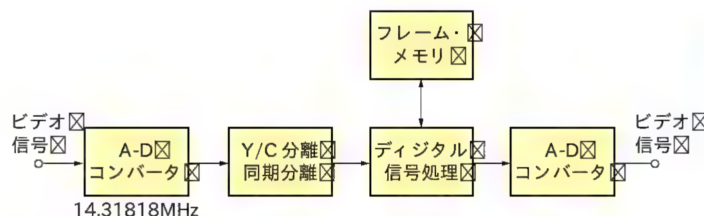


図3　一般的な画像信号処理のブロック図

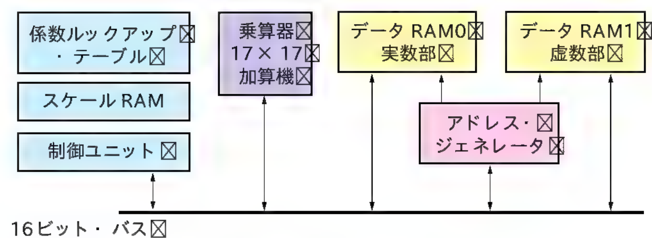


図5　ZR34161(ZORAN)内部ブロック概要

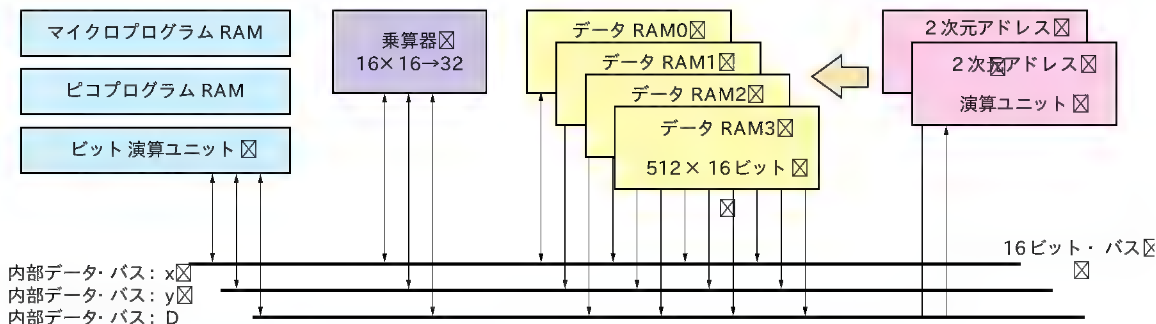


図4　DSP-I(HD81831，日立)内部ブロックの概略

New Products —— IFおよびRF周波数の入力レンジに対応した広帯域2チャネル・ベクトル乗算器
アナログ・デバイセズ社は，業界初のゲインおよび位相制御用の2チャネル・ベクトル乗算器「ADL5390」を発売した．これまで6個のディスクリート部品が必要であったワイヤレス・インフラストラクチャ機器のゲインおよび位相の制御を，一つのADL5390で行うことができる．

表4　代表的なMP3デコーダ/エンコーダLSIのメーカと型番

メーカ	型　番
MICRONAS社	MAS3507D
	MAS3587F
VLSI社	VS1001/VS1011/VS1002
ST-Microelectronics社	STA013/STA014/STA015

5000万曲に達したといわれている．

MP3の権利関係や音楽著作権の問題など，解決されるべき問題は多くあるにせよ，もはやこの流れは後戻りすることはできないだろう．通常，MP3のエンコードおよびデコードはPCで実行するが，ポータブル・オーディオ用途のために非常にコンパクトな録音/再生装置が各社より販売されている．その中身はワンチップ化されたMP3デコーダ/エンコーダ用LSIと，ごくわずかな周辺回路で構成されている．代表的なデコーダのおもなメーカおよび型番を表4に示す．MP3の原理および技術的な側面は本誌でも過去何度か取り上げているので，ここでは触れないでおく．

これらのLSIは，MP3デコード/エンコード・エンジンに特化したDSPコアとソフトウェア，およびインターフェースから構成されている．MICRONAS社とVLSI社のLSIは，外部よりプログラムをダウンロードすることもできる．これによりユーザのオリジナルのアルゴリズムで動作させたり，特殊効果を付加するというようなことも可能である．図6にVLSI社のVS1001のブロック図を示しておく．

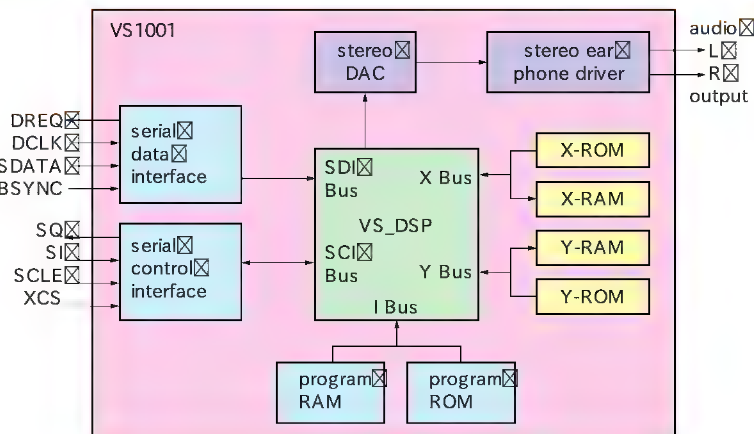


図6　VS1001（VLSI社）のブロック図

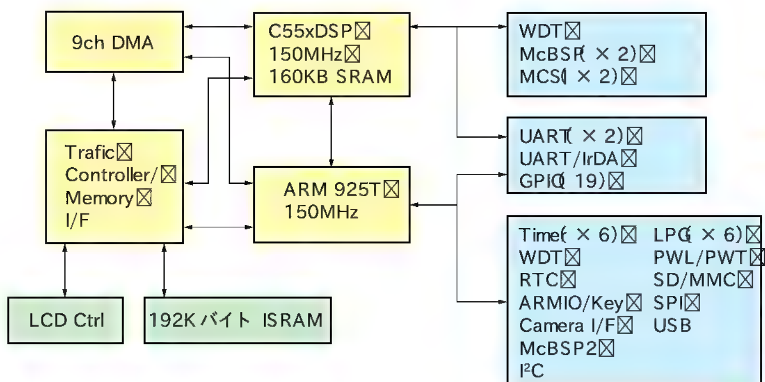


図7　OMAP5910（TI社）ブロック図

TI社では，MP3プレーヤとそのほかのポータブル・プレーヤをインターネット・オーディオ・プレーヤと呼び，TMS320C5000シリーズをターゲットにしてIPコアを提供している．このインターネット・オーディオIPを利用することで，よりフレキシブルなMP3プレーヤ・システムの開発時間の短縮が図れる．インターネット・オーディオIPで利用できるMP3を含むオーディオ規格および機能は，以下のように多くの中から選択できるようになっている．

- AC-3
- MPEG MP3
- MPEG AAC
- Sony ATRAC 1/2
- TwinVQ
- AC-3 5.1ch Decoder Class-A
- Virtual Dolby Digital
- MPEG-1 Layer Ⅰ Encoder/Decoder
- MPEG-1 Layer Ⅱ Encoder/Decoder
- AC-3 2ch Encoder（DDCE）Class A
- Liquid Audio SP3
- Lucent ePAC
- Qdesign QDMC
- System（Play, Stop, FF, Rew, Repeat）
- EQ（5-band）/Volume Control（Stereo）
- Sample Rate Converter（Stereo）

● 携帯電話――DSPコア＋RISCコア

MP3ポータブル・オーディオは，小型で高性能，そして低消費電力なDSPの開発が可能となったことにより，製品化された．この流れをさらに押し進めたものが，携帯電話やPDA，ディジタル・カメラなどの携帯型のディジタル機器である．これらの機器にはオーディオのみならず，画像処理の機能も要求されるため，より高機能なDSPの開発が不可欠となった．この目的のためTI社が投入した製品が「OMAP」というアプリケーション・プロセッサとも呼ばれるDSPである（図7）．

OMAPは，TI社の「TMS320C55x」DSPコアに加えて，ARM社のRISCコアを備えた，デュアル・コアのDSPである．これにより，Windows CEのような汎用OSが受け持つ，

- USBクライアント/ホスト・コントロール
- MMC-SDサポート

New Products――日本ラディシス，エーアイコーポレーションと協業してOS-9版RTFilesを提供
日本ラディシス（株）と（株）エーアイコーポレーションは，協業関係を結び，組み込みRTOSであるOS-9に組み込み用電源断対応ファイル・システム「RTFiles」を移植した製品を，日本国内および周辺地域で提供していく．

表5　代表的な次世代DSP

メーカ	シリーズ	型　番
Analog Devices社	Blackfin シリーズ	ADSP-BF533～ADSP-BF533
Freescale Semiconductor 社	MSC711x シリーズ	MSC7110～MSC7116
ルネサステクノロジ	SH-Mobileシリーズ	SH7290
TI社	C55x シリーズ	TMS320VC5501～TMS320VC5510

●Bluetoothインターフェース

以上のようなホストCPU機能と，通信，音声，画像処理といったリアルタイム処理機能を最適に振り分けることが可能となっている．OMAPは第2世代，第2.5世代，および第3世代の主要ワイヤレス通信プロトコルをサポートすることから，Nokia社やEricsson社，ソニーなどの主要携帯電話メーカが採用している．

次世代DSP
――キー・テクノロジはマルチメディア処理

ディジタル・ビデオ・カメラやDVDレコーダ/プレーヤ，第3世代の携帯電話などのマルチメディア装置の普及で，これらの要求に応えられるDSPの開発が急がれるようになってきた．つまり，キー・テクノロジは動画の圧縮/伸長技術にある．とくにトレンドはMPEG-4の処理を高速に実行するDSPの開発に向かっている．

また，上記以外にも，大容量光ディスク，双方向通信デバイスなどに使用するためのDSP開発競争は日々激化している．表5に各社が次世代DSPとして発表したものを示しておく．

次世代のマルチメディアの中心的役割を担う動画圧縮/伸長技術にとってコア・テクノロジとなるDSPは，これからますます発展の度合いが加速されることが予想され，目を離すことができなくなってきた．

参考文献

(1) Bernard Gold and Charles M. Rader : *Digital Processing of Signals*, McGraw-Hill, Inc. 1969

おおひさ・のぶひろ　ドリームコンピュータ(株)

TECH I Vol.2　好評発売中

ディジタル信号処理とDSP
パソコンによるシミュレーションとDSPプログラミング

三上 直樹 著　B5判 232ページ　CD-ROM付き
定価2,200円(税込)
ISBN4-7898-3313-5



パソコンの性能向上により，従来は不可能に思えたことが実現できるようになりました．本書で紹介するディジタル信号処理のシミュレーションもその一つです．ディジタル信号処理はその実践が数式で示されることが多く，直感的に理解しにくい分野です．そこで本書では，パソコン上でシミュレーションプログラムを作り，ディジタル信号処理が目や耳で理解できることを目指しました．

また，テキサス・インスツルメンツ社が販売しているDSPスタータキットを使い，実際のDSP(ディジタルシグナルプロセッサ)を用いたプログラミングについても詳細に解説しました．したがって，本書により，ディジタル信号処理の考え方とプログラミング方法が完璧にマスターできるようになります．

第1部　パソコンによるシミュレーション
第1章　信号処理シミュレーションの準備と標本化定理
第2章　ディジタルフィルタの基礎 ―― 移動平均
第3章　積分回路とIIRフィルタ
第4章　ディジタルフィルタの伝達関数とインパルス応答
第5章　適応フィルタと適応線スペクトル強調器
第6章　離散的フーリエ変換とFFT

第2部　DSPプログラミング
第7章　固定小数点DSP TMS320C5xとスタータキットの概要
第8章　TMS320C5xプログラミングの基礎
第9章　DSPスタータキットによるディジタルフィルタの実現
第10章　DSKスタータキットによる信号発生とその応用
第11章　浮動小数点DSP TMS320C3xとスタータキットの概要
第12章　DSK3xスタータキットによる信号発生とその応用
第13章　C言語によるDSK3xのためのディジタル信号処理プログラミング
第14章　PCと協調して動作するディジタル信号処理プログラム

CQ出版社　〒170-8461 東京都豊島区巣鴨1-14-2　販売部 TEL.03-5395-2141　振替 00100-7-10665

New Products――中小オフィス向けVoIPソリューション，IP-PBX・ゲートウェイ
日立電線(株)は，VoIP市場への本格参入を図るために，200ユーザ以内の中小規模オフィスでIP電話への移行をスムーズに行えるIP-PBX・ゲートウェイ「AquaVocal」シリーズを開発した．同製品により，投資を抑えながら段階的にオフィスのVoIP化を推進することが可能になる．

2

ディジタル・フィルタの基礎と設計，実装

FIR フィルタの設計と実現方法

本章では，ディジタル信号処理の中でも重要なディジタル・フィルタについて，基礎に焦点を置いて解説する．それを踏まえて，実際に FIR フィルタの設計を行い，設計したフィルタの実装方法についても述べる．
（編集部）

三上 直樹

ディジタル信号処理は情報通信をはじめとして広い分野で使われており，今日の IT 化社会を支えるインフラとして重要な役割を担っている．ディジタル・フィルタは，そのディジタル信号処理の中でも重要な位置を占めている．

そこで，本章ではディジタル・フィルタの考え方から出発し，その設計法や DSP への実装についての解説を行う．なお，ディジタル・フィルタで扱う信号にはいろいろな種類のものがあるが，この章では 1 次元で，時間とともに変化する信号，つまり時間信号を扱うディジタル・フィルタに限定して話を進める注1．

1 ディジタル・フィルタの考え方

ディジタル・フィルタというと難しいものだという先入観があるかもしれないが，実はそれほど難しいものではない．そこで，例を示しながらそれほど難しいことを行っているのではないということを説明する．

● 移動平均

図 1（a）には，東京都の 2004 年 1 月 1 日から 5 月 31 日までの最高気温[1]の変化を示す．この図を見ると，グラフが上下に激しく変動しているため，気温の大ざっぱな変化のようすを把握しにくい．このような場合，時系列データの統計的な処理の手法としてよく使われているものの一つが移動平均（moving average）を行う方法である．移動平均は従来から使われている手法であるが，この方法を図 2 に示す．つまり，平均の計算に使うデータの範囲を一つずつ時間軸に沿って移動しながら平均の計算を行っていくのである．

図 1（a）のデータに対して，平均を行う数を 5として移動平

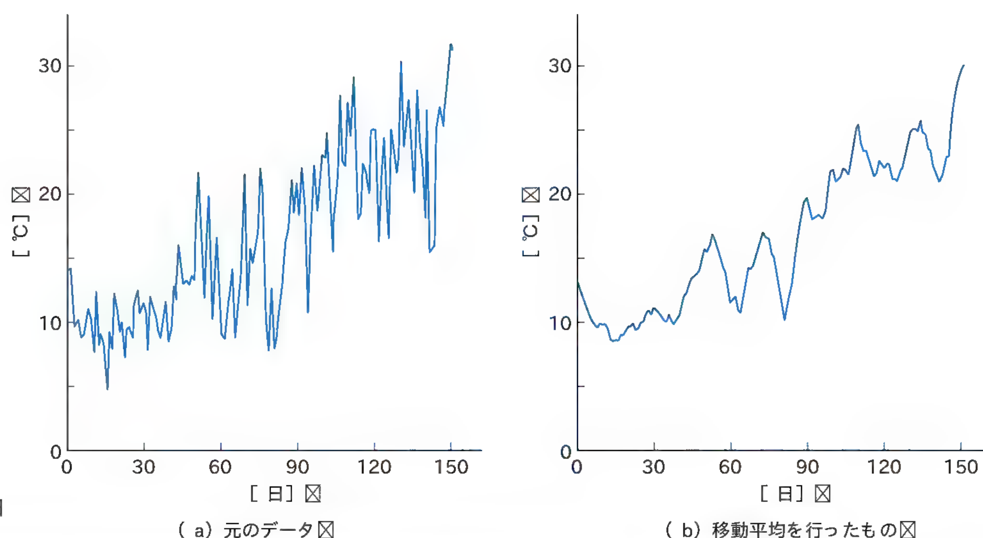


図 1
東京都の 2004 年 1 月 1 日から 5 月 31 日における毎日の気温の変化

注 1：時間信号に限定するのは，用語が煩雑になるのを避けるためである．したがって，時間信号というのは本質的な制約ではないため，この章の話は時間信号以外，たとえば画像信号などにも拡張することができる．

COLUMN-01

標本化

ディジタル・フィルタで処理を行う場合，対象となる信号の多くはアナログ信号である．アナログ信号は時間について連続な信号であるが，ディジタル・フィルタで扱う信号は時間について離散的な信号である．したがって，ディジタル・フィルタで処理を行う場合は，アナログ信号に対して等間隔に標本化（sampling）という操作を行って，離散的信号に変換してから処理を行う．この間隔を標本化間隔または標本化周期と呼び，その逆数を標本化周波数と呼ぶ．このとき重要なのが標本化定理である．この定理から次のことが導き出される．

アナログ信号が周波数0～F_0の範囲に帯域制限されている場合，つまり0～F_0以外の周波数成分が含まれていない場合，標本化周波数F_Sは次の式を満足しなければならない．

$$F_S \geq 2F_0 \quad \cdots\cdots (A.1)$$

この条件が満足されていれば，標本化を行う前のアナログ信号と，標本化された後の離散的信号は等価であることが保証される．

一方，この条件が満足されない場合には，エイリアシング（aliasing）と呼ばれる現象を生じる．その結果，元のアナログ信号に本来は含まれていない周波数成分が標本化後の離散的信号に現れることになる．

均を行った結果を図1（b）に示す．この結果から，最高気温の大ざっぱな変化のようすがよくわかるようになる．

ところで，データの変動が激しいということは，見方を変えれば高い周波数成分が多く含まれているということができる．逆にデータが滑らかに変動しているということは，高い周波数があまり含まれていないということができる．したがって，図1（a），（b）を比較すると，（b）のほうが滑らかであるから，移動平均は高い周波数成分を減少させる働きを持っていることがわかる．このような働きを持ったシステムを，信号処理の世界では低域通過フィルタ，またはローパス・フィルタ（low pass filter）と呼んでいる．

ここで行っている移動平均を式で表すと次のようになる．

$$y[n] = \frac{1}{5}(x[n-2] + x[n-1] + x[n] + x[n+1] + x[n+2])$$
$$= \frac{1}{5}\sum_{m=-2}^{2} x[n-m] \quad \cdots\cdots (1)$$

このような式は差分方程式（difference equation）と呼ばれている．この式で，nは任意の整数である．したがって，この式はnを一つずつ増加させながら順に平均の操作を行っていくことを表している．つまり，n=0を計算の開始点とすると，式（2）のように，次々に計算を行っていくことを表している．

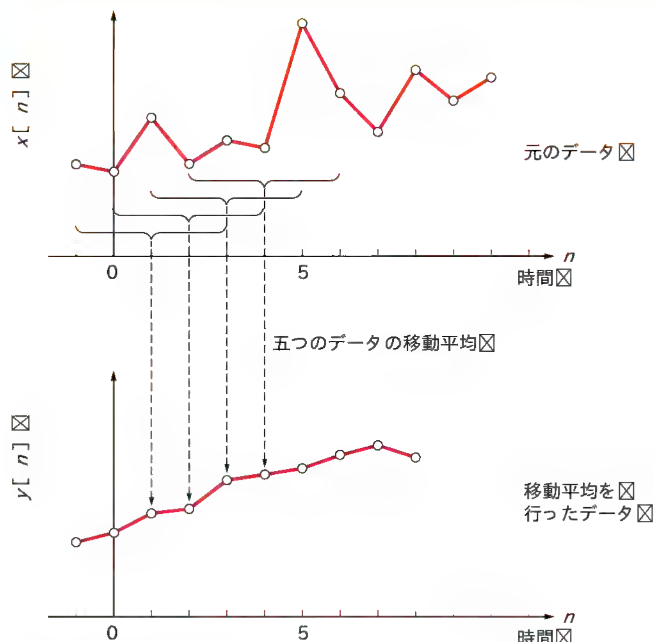


図2　移動平均のようす

$$n=0 \text{ のとき } y[0] = \frac{1}{5}(x[n-2]+x[n-1]+x[n]+x[n+1]+x[n+2])$$
$$n=1 \text{ のとき } y[1] = \frac{1}{5}(x[n-1]+x[n]+x[n+1]+x[n+2]+x[n+3])$$
$$n=2 \text{ のとき } y[2] = \frac{1}{5}(x[n]+x[n+1]+x[n+2]+x[n+3]+x[n+4])$$
$$n=3 \text{ のとき } y[3] = \frac{1}{5}(x[n+1]+x[n+2]+x[n+3]+x[n+4]+x[n+5])$$
$$\vdots \qquad \vdots \qquad \cdots\cdots (2)$$

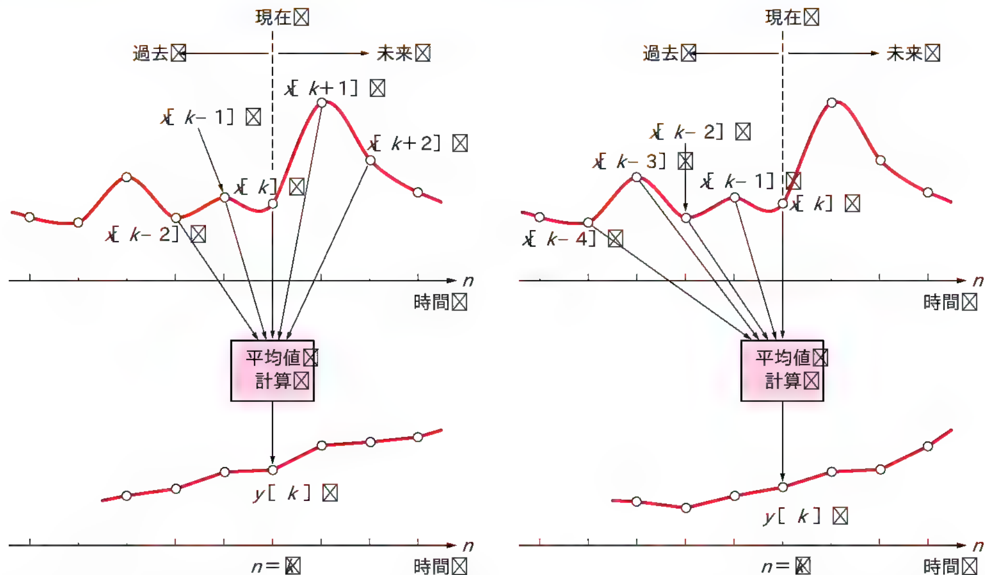
ここまでの説明で，移動平均の操作が低域通過フィルタの操作と同じことであるということは，感覚的には理解できたと思う．しかし，本当に低域通過フィルタと同じなのかということを確かめるためには，式（1）の操作を行うシステムの持つ周波数特性を計算する必要がある．これについては次項の周波数応答のところで説明する．

● リアルタイム処理による移動平均

前の項で移動平均の方法について説明したが，このように，あらかじめデータが得られていて，そのデータに対して後でまとめて移動平均を行うという場合は何も問題はない．しかし，リアルタイム処理で移動平均を行う場合には不都合が生じる．このことを，図3を使って説明する．

図3にはn=kという時刻における移動平均の値を計算するようすを示している．図3（a）では現在の値$y[k]$を計算するために，$x[k+1]$，$x[k+2]$というデータも使っているが，これは未来のデータであるから，n=kの時刻においては，まだ計算に使うことができない．したがって，このような方法はリアルタ

New Products ――アライドテレシス，最長100m最大100Mbpsの通信を実現する光無線LAN装置を発売
アライドテレシス（株）は，赤外線直接輝度変調方式を用いて最長100mの距離（都市部全天候）を最大100Mbpsで結ぶことのできる光無線LAN「CentreCOM WP101」を発売した．複数の建物に分散しているオフィス間や公道などで隔てられた建物間での使用に適している．



(a) 未来のデータを使うのでリアルタイム処理不可能　(b) 未来のデータを使わないのでリアルタイム処理可能

図3　リアルタイム処理による移動平均の説明（時刻 $n=k$ の $y[k]$ を計算するようす）

イム処理としては実現できないことになる．そこで，図3(b)のように，現在および過去のデータを使って平均を行うようにすれば，この処理はリアルタイムで行うことが可能になる．

図3(b)の処理を差分方程式で表現すると次のようになる．

$$y[n]=\frac{1}{5}(x[n-4]+x[n-3]+x[n-2]+x[n-1]+x[n])$$

$$=\frac{1}{5}\sum_{m=0}^{4}x[n-m] \quad \cdots\cdots (3)$$

● ブロック図

式(1)や式(3)の差分方程式を図で表す場合に，よく使う方法としてブロック図(block diagram)とシグナル・フロー・グラフ(signal flow graph)があるが，ここではブロック図を使うことにする．

図4にはブロック図の要素を示す．これらを使うと式(3)に対応するブロック図は図5のようになる．この図には各部分に現れる信号も示しておいたので，各要素との対応関係が理解できると思う．

● 移動平均の拡張

式(3)に分配法則を適用すると，次のように書くことができる．

$$y[n]=\frac{1}{5}x[n-4]+\frac{1}{5}x[n-3]+\frac{1}{5}x[n-2]+\frac{1}{5}x[n-1]+\frac{1}{5}x[n]$$

$$=\sum_{m=0}^{4}\left(\frac{1}{5}x[n-m]\right) \quad \cdots\cdots (4)$$

この式に対応するブロック図は図6のようになる．図6のシステムと図5のシステムに同じ入力信号を与えると，当然ながら同じ信号を出力する．

ところで，図6のシステムでは乗算器の係数の値はすべて1/5という同じ値になっているが，個々の乗算器に異なる係数

記号	記号	機能
$x[n]$ →▷ a → $y[n]$	乗算器	入力を a 倍したものを出力する $y[n]=ax[n]$
$x_1[n]$ →(+)→ $y[n]$，$x_2[n]$	加算器	二つの入力の和を出力する $y[n]=x_1[n]+x_2[n]$
$x_1[n]$ →(+, −)→ $y[n]$，$x_2[n]$	減算器	二つの入力の差を出力する $y[n]=x_1[n]-x_2[n]$
$x[n]$ →• → $y_1[n]$，$y_2[n]$	分岐点	同じ信号を2か所へ出力する $y_1[n]=x[n]$，$y_2[n]=x[n]$
$x[n]$ → z^{-1} → $y[n]$	遅延器*	入力を1サンプル分遅らせて出力する $y[n]=x[n-1]$

*遅延器は T や D のように表す場合もある

図4　ブロック図の要素

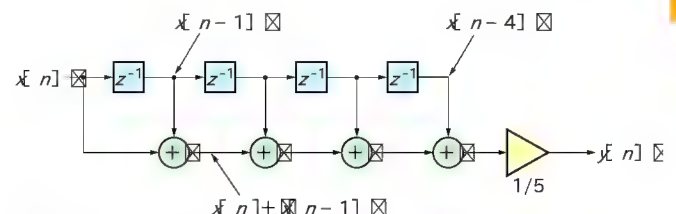


図5　式(3)に対応する移動平均のブロック図

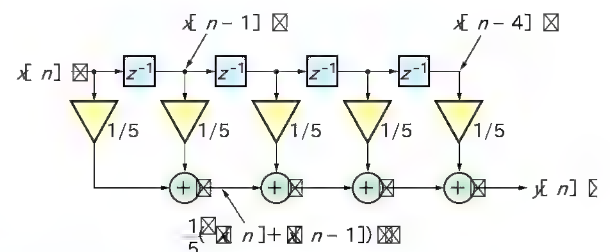


図6　式(4)に対応する移動平均のブロック図

New Products ――アルプス電気，IEEE802.11 通信用モジュール 3 種類を発売
アルプス電気(株)は，IEEE802.11a/b/g 通信用モジュール「UGJZ1」「UGJZ2」シリーズおよび IEEE802.11b通信用モジュール「UGGZ」シリーズを発売した．UGJZ1/UGJZ2は，WEP/TKIP/AES 搭載によりセキュリティを充実させた Super A/G 対応タイプである．UGJZ1シリーズは，基板片面のみ実装したもの．

メモ

z変換の性質

z変換の操作を$\mathcal{Z}\{\cdot\}$という記号で表すものとする．

1. 線形性

$$\mathcal{Z}\{ax_1[n]+bx_2[n]\}=a\mathcal{Z}\{x_1[n]\}+b\mathcal{Z}\{x_2[n]\}$$ ，a，bは定数

2. 時間軸上のシフト

$$\mathcal{Z}\{x[n-k]\}=z^{-k}\mathcal{Z}\{x[n]\}$$

を与えることも可能である．異なる係数を与えた場合，この平均は単純な平均ではなく重み付き平均になる．それぞれの重みをh_0，h_1，…，h_4とすると，式(4)は次のようになる．

$$y[n]=\sum_{m=0}^{4}h_m x[n-m] \quad \cdots\cdots (5)$$

この式は，ディジタル･フィルタの一種であるFIR(Finite Impulse Response)フィルタ[注2]を表す差分方程式と同じものである．

式(5)では，計算に使う入力信号は$x[n]$，$x[n-1]$，…，$x[n-4]$の5個であるが，$x[n-M]$まで使うものとすると，次のようになる．

$$y[n]=\sum_{m=0}^{M}h_m x[n-m] \quad \cdots\cdots (6)$$

この式が，FIRフィルタを表す一般的な差分方程式になる．また，この式のMをこのフィルタの次数[注3]という．なお，この式のh_mは重みを表すが，一般的にはフィルタの係数と呼ばれている．

2 ディジタル・フィルタの周波数特性と移動平均

フィルタにはいろいろな特性を持つものがあるが，よく使われるフィルタは，周波数選択性を持ったものである．つまり，ある周波数成分はよく通し，別の周波数成分はあまり通さないといった特性を持つものである．このようなフィルタは線形フィルタ[注4]と呼ばれる．また，フィルタにはその性質が時間とともに変化したり[注5]，入力信号の性質により変化する[注6]というものもあるが，ここでは周波数選択性を持ち，その性質が時間や入力信号の性質では変化しないフィルタを扱うことにする．

フィルタの周波数特性はフィルタの伝達関数から求めることができる．そこで，最初に伝達関数について説明し，その後，周波数特性を表す関数である周波数応答について説明する．

● 伝達関数

伝達関数は差分方程式から求めることができる．その際にはz変換の知識が必要となるが，z変換のすべてについて知っている必要はなく，メモに示す二つの性質だけを理解しておけば十分である．なお，参考までにz変換の定義について，コラム2に示す．

あるシステムの入力信号を$x[n]$，出力信号を$y[n]$とすると，そのシステムの伝達関数$H(z)$は次のように定義される．

$$\text{伝達関数：}\ H(z)=\frac{\mathcal{Z}\{y[n]\}}{\mathcal{Z}\{x[n]\}} \quad \cdots\cdots (7)$$

そこで，入出力の関係が，式(6)の差分方程式で表されるシステムの伝達関数を求めてみよう．z変換の性質を考慮して式(6)の両辺をz変換すると次のようになる．

$$\mathcal{Z}\{y[n]\}=\sum_{m=0}^{M}h_m\left(\mathcal{Z}\{x[n-m]\}\right)$$
$$=\sum_{m=0}^{M}h_m z^{-m}\left(\mathcal{Z}\{x[n]\}\right) \quad \cdots\cdots (8)$$

したがって，式(6)で表されるFIRフィルタの伝達関数$H(z)$は次のようになる．

$$H(z)=\sum_{m=0}^{M}h_m z^{-m} \quad \cdots\cdots (9)$$

● 周波数応答

伝達関数はzという変数の関数であるが，伝達関数において

$$z=\exp(j\omega T) \quad \cdots\cdots (10)$$

と置き換えたものが周波数応答(frequency response)[注7]である．この式で，Tは標本化の間隔で，標本化周波数の逆数に等しい．また，jは虚数単位で$j=\sqrt{-1}$，ωは角周波数である．

したがって，式(9)に対応する周波数応答$H(\omega)$[注8]は次のようになる．

$$H(\omega)=\sum_{m=0}^{M}h_m \exp(-j\omega Tm) \quad \cdots\cdots (11)$$

この値は複素数になる．これを次のような極形式で表現すると，

注2：ディジタル・フィルタは大きく二つに分類でき，一つがFIRフィルタ，もう一つがIIR(Infinite Impulse Response)フィルタである．この章ではFIRフィルタのみを取り上げる．
注3：次数がM次であるFIRフィルタの係数の数は$M+1$個になる．
注4：線形フィルタに対して非線形フィルタも存在するが，このタイプのフィルタはその性質を周波数選択性では記述できない．
注5：このようなフィルタは時変フィルタ(time-variant filter)と呼ばれる．これに対して時間とともに特性が変化しないフィルタは時不変フィルタ(time-invariant filter)と呼ばれる．
注6：このようなフィルタは適応フィルタ(adaptive filter)と呼ばれる．
注7：周波数応答関数と呼ばれる場合もある．
注8：$H(e^{j\omega T})$と書くこともある．
注9：入力信号の振幅に対する出力信号の振幅の比が周波数の変化とともにどのように変化するのかを表したもの．

New Products ── PCやLAN接続HDDに保存したコンテンツもTVで視聴できるDVDドライブ搭載ネットワーク・メディア・プレーヤ
(株)バッファローは，DVDドライブを搭載し，PCやLAN接続HDDに保存したコンテンツもネットワーク経由で視聴可能なネットワーク・メディア・プレーヤ「LinkTheater」の新製品として，DivXに正式対応，対応ファイル拡張などの機能アップを施した「PC-P2LAN/DVD」を発売した．価格は￥26,565．

COLUMN-02

z変換とラプラス変換

アナログ信号を標本化して得られる離散的信号を$f[n]$とすると，そのz変換（z-transform）である$F(z)$は次の式で定義される．

$$F(z)=\sum_{n=0}^{\infty}f[n]z^{-n} \quad \cdots\cdots (B.1)$$

これとは逆に，z変換$F(z)$が与えられたとき，それに対応する離散的信号$f[n]$を求めるには逆z変換（inverse z-transform）を使う．その定義は次のようになる．

$$f[n]=\frac{1}{2\pi j}\oint_C F(z)z^{n-1}dz \quad \cdots\cdots (B.2)$$

この式で，Cは積分路の閉曲線で，積分は左回り（counter clockwise）に行う．通常，この積分路は被積分関数$F(z)z^{n-1}$のすべての極を囲むようにとる．

z変換は，アナログ回路素子によるシステムを解析する場合によく使うラプラス変換（Laplace transform）の特別な場合に相当する．ある信号$f(t)$に対して，そのラプラス変換$F(s)$は次の式で定義される．

$$F(s)=\int_0^{\infty}f(t)\exp(-st)dt \quad \cdots\cdots (B.3)$$

そこで，離散的信号$g[n]$のラプラス変換を求めてみる．ところで$g(t)$を標本化間隔Tで標本化して得られた離散的信号$g[n]$は$n\geq 0$において，ディラックのデルタ関数$\delta(t)$を使うと，次のように表現できる．

$$g[n]=\sum_{n=0}^{\infty}g(t)\delta(t-Tn) \quad \cdots\cdots (B.4)$$

したがって，離散的信号$g[n]$のラプラス変換$G(s)$は次のようになる．

$$G(s)=\int_0^{\infty}\left(\sum_{n=0}^{\infty}g(t)\ (t-Tn)\right)\exp(-st)dt$$
$$=\sum_{n=0}^{\infty}\left(\int_0^{\infty}\{g(t)\exp(-st)\}\delta(t-Tn)\,dt\right)$$
$$=\sum_{n=0}^{\infty}g(nT)\exp(-sTn)$$

この式で

$$z=\exp(sT) \quad \cdots\cdots (B.5)$$

とおくと，次のようになる．

$$G(z)=\sum_{n=0}^{\infty}g(nT)z^{-n} \quad \cdots\cdots (B.6)$$

通常，標本化を行って得られる信号は，Tを省略して表現するので，$g(nT)$と$g[n]$は同じものである．そうすると，式（B.6）はz変換の定義である式（B.1）とまったく同じになる．

$$H(\omega)=|H(\omega)|\exp(j\theta(\omega)) \quad \cdots\cdots (12)$$

となる．ここで，$|H(\omega)|$は振幅に関する周波数特性[注9]（以下では単に振幅特性と呼ぶ），$\theta(\omega)$は位相に関する周波数特性[注10]を表す．

なお，式（11）でωの代わりに$\omega+2\pi n/T$（n：整数）を代入しても$H(\omega)$は同じ値になることから，周波数応答$H(\omega)$はω軸上で$2\pi/T$を周期とする周期関数になることがわかる．

● 単純な移動平均の振幅特性

以上のことから，単純な移動平均，つまり式（6）のh_mがすべて等しい場合［$h_m=1/(M+1)$，$m=0, 1, \cdots, M$］の周波数応答は次のようになる．

$$H(\omega)=\frac{1}{M+1}\sum_{m=0}^{M}\exp(-j\omega Tm) \quad \cdots\cdots (13)$$

この式から振幅特性を求めると次のようになる．

$$|H(\omega)|=\frac{1}{M+1}\cdot\left|\frac{\sin\frac{(M+1)\omega T}{2}}{\sin\frac{\omega T}{2}}\right| \quad \cdots\cdots (14)$$

図7には，M=1，2，6，20の場合の振幅特性を示す．この図で横軸は標本化周波数F_sを基準にして目盛りを付けている[注11]．この図から，移動平均の操作を行うことにより高い周波数成分を減衰させられることがわかる．つまり，移動平均の操作は低域通過フィルタの操作と同じであることがわかる．

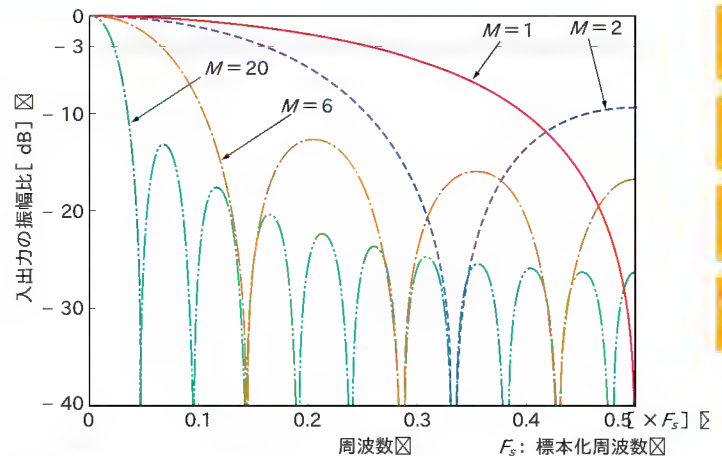


図7 移動平均の振幅特性

なお，フィルタの特性を示すパラメータの一つとして遮断周波数（cutoff frequency）がある．遮断周波数の定義はいろいろあるが，よく使われるのは出力信号の振幅が基準の周波数（低域通過フィルタの場合は周波数0Hz）における値に対して，3dB低下する周波数を遮断周波数とするものである．この定義を使

注10：入力信号と出力信号の位相差が周波数の変化とともにどのように変化するのかを表したもの．
注11：横軸が0.5F_sまでなのは，離散的信号の場合，扱うことのできる最高の周波数は標本化定理から規定され，その値が0.5F_sだからである．

Information ――ザイリンクスとUMC，トリプル酸化膜テクノロジを90nmプロセスに適用したFPGAを量産開始
ザイリンクス社と台湾UMCは，90nmプロセスとトリプル酸化膜テクノロジを採用したFPGA製品の量産開始を発表した．これにより Virtex-4プラットホーム FPGAファミリのダイナミックおよびスタティック消費電力を従来製品に比べ50％低減した．

うと，図7の振幅特性から，遮断周波数は表1のようになる．

以上の結果から，移動平均の操作では M が増加するにしたがって遮断周波数が低くなることがわかる．

次に，図7を阻止域の特性という観点で考えてみる．阻止域とは，入力の振幅に対して出力の振幅が十分小さくなるような帯域のことである．図7を見ると，阻止域にいくつかのピークが生じている．たとえば M=6の場合，これらのピークの周波数と大きさは表2のようになっている．

表2から，阻止域であっても，周波数によっては出力の振幅があまり減衰しない場合もあるということがわかる．M=6のとき，最悪で減衰量が約12.7dBにとどまっている．減衰量の最悪値は M を増加してもそれほど増加せず，M=100の場合で約13.3dBとなり，M をそれ以上大きくしても約13.3dBという値はほぼ一定となる．

以上のことから，移動平均は低域通過フィルタとしての働きは持っているが，阻止域での特性はあまり良いとはいえないことがわかる．また，遮断周波数を任意に設定できないこともわかる．それでは，その対策はどうしたらよいのかということになるが，これについては次の項で説明する．

3 移動平均から一般的なFIRフィルタへ

前の項ではフィルタの周波数特性の計算方法を説明し，移動平均に対応する振幅特性を求めた．その結果，移動平均では阻止域の減衰量をあまり大きくはできず，また遮断周波数を任意に設定できないこともわかった．それではこれらの点を改善するためにはどうしたらよいかということが問題になる．

一般的なFIRフィルタを表す差分方程式は既に出てきたが，再度以下に示す．

$$y[n]=\sum_{m=0}^{M} h_m x[n-m] \quad \cdots\cdots (15)$$

表1　M を変えたときの遮断周波数の変化

M	遮断周波数
1	$0.25F_s$
2	$0.16F_s$
6	$0.06F_s$
20	$0.02F_s$

表2　M=6の場合における阻止域のピークの大きさ

周波数	ピークの大きさ（dB）
$0.21F_s$	－12.7
$0.35F_s$	－16.0
$0.5F_s$	－16.9

移動平均の場合は，この式の係数 h_m（m=0, 1, …, M）がすべて同じ値であった．しかし，この値はそれぞれ異なっても差し支えない．そこで，以下の式（16b），（17b）で示される2通りの h_m を使ってみる．ただし M は偶数とする．なお，いずれの場合も $\sum_{m=0}^{M} h_m'$ で割り算を行っているのは，周波数0における入出力の振幅比が1になるようにするためである．

1）三角形状の係数

$$h_m' = h_{M-m}' = m+1,\quad m=0, 1, \cdots, M/2 \quad \cdots\cdots (16a)$$

$$h_m = \frac{h_m'}{\sum_{m=0}^{M} h_m'},\quad m=0, 1, \cdots, M \quad \cdots\cdots (16b)$$

2）cos関数を使って表現される係数

$$h_m' = 1+\cos\left(\frac{2\pi(m-M/2)}{M+2}\right),\quad m=0, 1, \cdots, M \quad \cdots\cdots (17a)$$

$$h_m = \frac{h_m'}{\sum_{m=0}^{M} h_m'},\quad m=0, 1, \cdots, M \quad \cdots\cdots (17b)$$

M=10の場合について，式（16b），（17b）で示される2通りの h_m を図8に示す．これらの係数を用いた場合と係数が同じ場合の振幅特性を計算すると図9のようになる．この図から，係数が同じ場合に比べて，三角形状の係数やcos関数を使って計算される係数のほうが，阻止域における減衰量の最悪値を大きくできることがわかる．このとき，遮断周波数と阻止域におけ

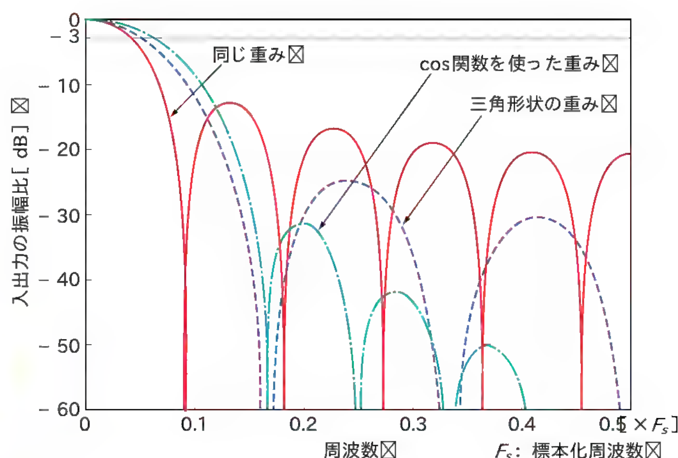
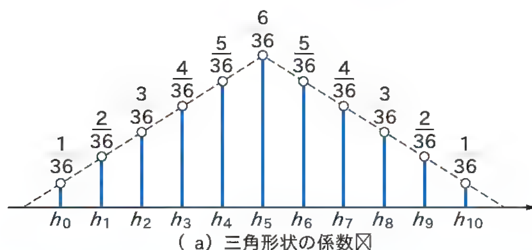
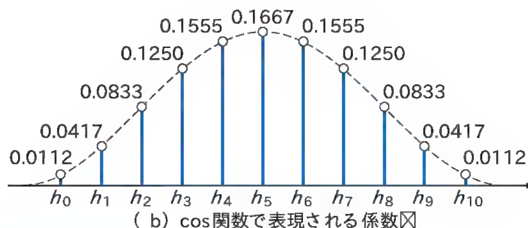


図9　重みを変えた場合の振幅特性（M=10）



（a）三角形状の係数



（b）cos関数で表現される係数

図8　フィルタの係数

Information ——日立，3次元構造のSRAMセルを試作
（株）日立製作所は（株）ルネサス テクノロジと共同で，3次元構造のSRAMセルの開発と動作の確認に成功した．メモリ・セル面積を従来に比べ1/3に削減することができ，45nmプロセス世代以降の，小型・高集積のSRAM実現に向けての基本構造として期待される．

表3　各種係数に対応するフィルタの特性（M=10の場合）

係数の種類	遮断周波数	阻止域における減衰量の最悪値（dB）
すべて同じ係数	$0.040F_s$	13.0
三角形状の係数	$0.054F_s$	24.9
cos関数による係数	$0.060F_s$	31.5

る減衰量の最悪値を表3に示す．

以上のことから，係数をくふうすれば阻止域における減衰量の最悪値を改善できることがわかる．実際にフィルタを実現する場合には阻止域における減衰量の最悪値だけではなく，遮断周波数をはじめとするそのほかの特性に基づいてフィルタの係数が決定されなければならない．そのような係数は試行錯誤的に見つけることもできるが，それでは非常に非効率的である．そこで，これを効率よく行う方法が必要となる．そのような方法は，一般にディジタル・フィルタの設計法と呼ばれており，いろいろな設計方法が発表されている．

次の二つの項では，FIRフィルタの設計法として代表的なものである，窓関数法とParks-McClellan法について説明し，その次の項では筆者の作成した設計プログラムを紹介する．

4 窓関数法によるFIRフィルタの設計法

この方法は任意の振幅特性および任意の位相特性のフィルタを近似するような係数を設計できる．しかし，ここでは式を具体的に示すため，理想的な低域通過フィルタを近似するための係数を求めるものとして話を進める．

● 低域通過フィルタの設計

図10に設計の概念を示す．この方法では，最初に近似するフィルタの周波数応答$G(\omega)$を決める．遮断角周波数をω_C，標本化間隔をTとすると，理想的な低域通過フィルタの周波数応答$G(\omega)$は$|\omega| \le \pi/T$の範囲のωに対して，次の式で与えられる．

$$G(\omega) = \begin{cases} 1 & (|\omega| \le \omega_C) \\ 0 & (\omega_C < |\omega| \le \pi/T) \end{cases} \quad \cdots\cdots (18)$$

これを図10（a）に示す．

ディジタル・フィルタの周波数応答は$2\pi/T$（$=\omega_S$，ω_S：標本化角周波数）を周期とする周期関数であることは2の中の，周波数特性に関する説明のところで示した．したがって，$G(\omega)$は以下のようにフーリエ級数展開することが可能である．

$$G(\omega) = \sum_{m=-\infty}^{\infty} g_m \exp(-jm\omega T) \quad \cdots\cdots (19)$$

この式の展開係数であるg_mは次の式で求めることができる．

$$g_m = \frac{T}{2\pi}\int_{-\pi/T}^{\pi/T} G(\omega)\exp(jm\omega T)\,d\omega,$$
$$m = \cdots, -1, 0, 1, 2, \cdots \quad \cdots\cdots (20)$$

$G(\omega)$は式（18）より，偶関数である．それを考慮し，さらに

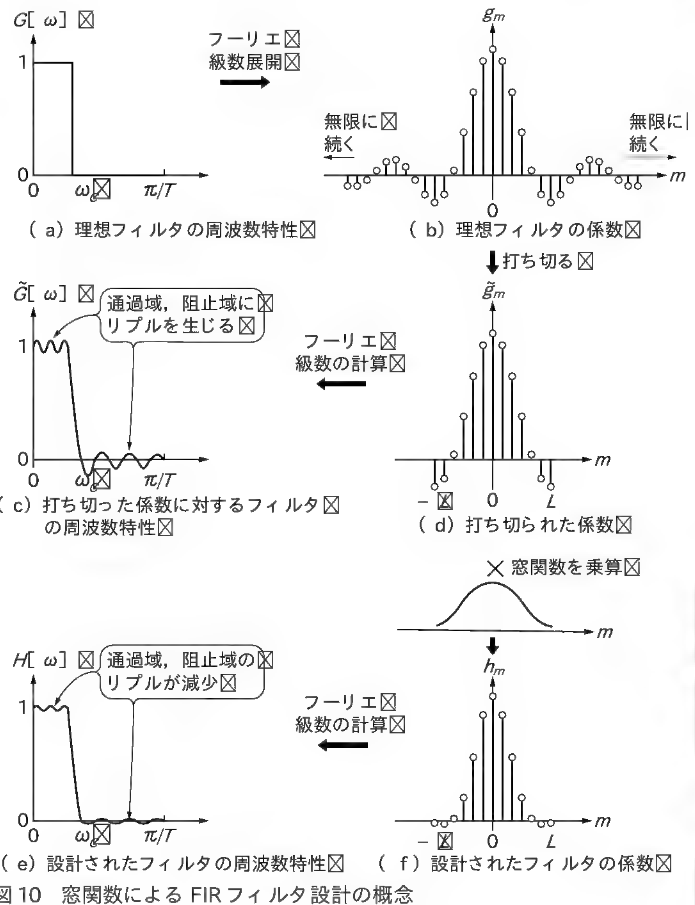


図10　窓関数によるFIRフィルタ設計の概念

式（18）を代入すると，式（20）は次のようになる．

$$g_m = \frac{T}{\pi}\int_0^{\pi/T} G(\omega)\exp(jm\omega T)\,d\omega$$
$$= \frac{T}{\pi}\int_0^{\omega_C} \cos(m\omega T)\,d\omega$$
$$= \frac{1}{m\pi}\sin(m\omega_C T),\quad m = \cdots, -1, 0, 1, 2, \cdots \quad \cdots\cdots (21)$$

このg_mは図10（b）のようになる．

一方，フィルタの周波数応答をフーリエ級数展開したときの展開係数の列はフィルタのインパルス応答に等しいことが知られている[2]．また，FIRフィルタでは，インパルス応答は，式（6）の差分方程式の係数h_mに等しい．

以上のことから，式（21）で求められるg_mは式（18）に示す周波数応答を持つフィルタの係数ということになる．しかし，式（19）は無限級数であるため，式（18）に示す周波数応答を実現するためには無限個のg_mを使わなければならない．ところが，式（21）の右辺は先頭に$1/m$の乗算があるため，図10（b）に示すように，$|m|$が大きくなるとg_mは振動しながら，その絶対値$|g_m|$は小さくなっていく．そこで，ある整数Lを決めて，$|m|<L$の項を切り捨てても$G(\omega)$を近似できることが予想される．

したがって，次の式で与えられる$\tilde{g}_m$をフィルタの係数とすることを考える．

News Flash ――インフィニオンとIBM，世界初の16MビットMRAM（磁気抵抗RAM）を開発
独インフィニオンテクノロジーズ社とIBM社は，共同で16MビットMRAM（Magnetoresistive Random Access Memory）をデモ公開した．このメモリ・チップは不揮発性で，これまで発表された中では最高の記憶密度をもつMRAMである．MRAMは既存技術に比べ高速かつ低消費電力であることが特徴．

$$\tilde{g}_m = \begin{cases} g_m, & |m| \leq L \\ 0, & |m| > L \end{cases} \quad \cdots\cdots (22)$$

これを図10(d)に示す．

ところが，式(19)で g_m の代わりに $\tilde{g}_m$ を代入して求めた周波数応答を $\tilde{G}(\omega)$ とすると，フーリエ級数に関する理論で明らかになっていることであるが，図10(c)に示すように，$\tilde{G}(\omega)$ の通過域や阻止域の特性に大きなリプル(ripple)[注12]を生じることになる．これでは，フィルタの特性としては好ましくない．

このリプルは，$\tilde{g}_m$ を m の関数と考え，さらに m を実数と仮定したとき，$|m|=L$ の箇所で $\tilde{g}_m$ が不連続[注13]になっているために生じたものである．そこで，この不連続性を減らすため，$\tilde{g}_m$ に対して，中央部は大きく，端に行くに従って徐々に小さくなるような重みを乗算する．この重みは窓(window)関数と呼ばれている．このようにすると図10(e)に示すように，通過域や阻止域の特性に生じるリプルを減らすことができるので，これをフィルタの係数とする．この窓関数を w_m とすると，求められるフィルタの係数 h_m は次のようになる．

$$h_m = \begin{cases} w_m \cdot \tilde{g}_m, & |m| \leq L \\ 0, & |m| > L \end{cases} \quad \cdots\cdots (23)$$

これを図10(f)に示す．この結果，フィルタの係数 h_m の個数は2L＋1個になる．

この方法で重要なのは，窓関数の選択である．窓関数はFFTでスペクトル解析を行う場合に使われ，ハニング(Hanning)窓，ハミング(Hamming)窓，ブラックマン(Blackman)窓がよく知られているが，これらの窓関数を使うと設計の際の自由度が減る．そこで，ここではカイザー(Kaiser)窓を使う．

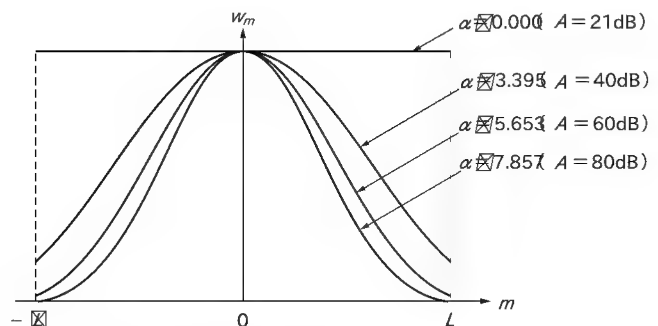


図11　種々のαに対するカイザー窓
かっこ内の数値は，式(25)でαを求める際に使ったAの値

● カイザー窓によるフィルタの設計[3]

カイザー窓を w_m とすると，次の式で与えられる．

$$w_m = \begin{cases} \dfrac{I_0\left(\alpha\sqrt{1-(m/L)^2}\right)}{I_0(\alpha)}, & |m| \leq L \\ 0, & |m| > L \end{cases} \quad \cdots\cdots (24)$$

この式で，$I_0(x)$ は0次の第1種変形ベッセル関数[注14](modified zeroth-order Bessel function of first kind)である．いくつかの α に対応するカイザー窓を図11に示す．なお，この図でかっこ内の A の値は，設計されるフィルタの阻止域における減衰量の最悪値をdBで表したもので，図12中の A に相当する．カイザー窓を表す式(24)には二つのパラメータ α，L が含まれているが，これらは設計されるフィルタの特性に関係する．

パラメータは，阻止域における減衰量(図12の A)で決まる量であり，その関係は次の式で表される．

$$\alpha = \begin{cases} 0.1102(A-8.7), & A \geq 50 \\ 0.5842(A-21)^{0.4} + 0.07886(A-21), & 21 < A < 50 \\ 0, & A \leq 21 \end{cases} \quad \cdots\cdots (25)$$

この式は実験的に求められたものなので，A に対応する α の値が多少異なる場合もある．

パラメータ L[注15]は，フィルタの遷移域の幅，つまり通過域から阻止域に移行する間の帯域の幅と関係がある．遷移域の幅 $\Delta\omega$ はカイザー窓を使って設計されるフィルタの振幅特性を表

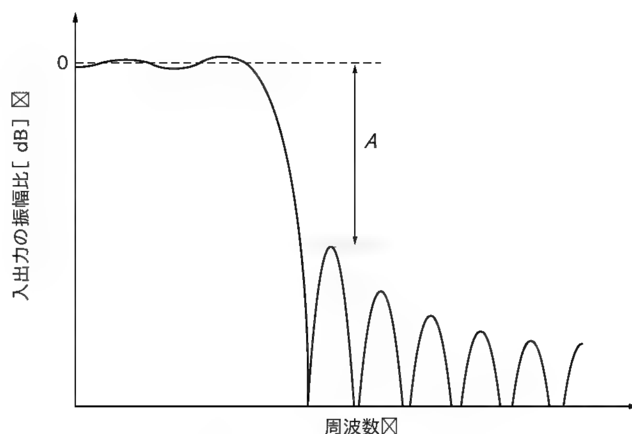


図12　阻止域における減推量の最悪値(A)の説明

注12：リプル(ripple)を生じるとは，波打つような状態になることを指す．

注13：$\tilde{g}_m$ そのものだけではなく，$\tilde{g}_m$ の導関数(高次の導関数も含む)も含めた不連続性．

注14：0次の第1種変形ベッセル関数 $I_0(x)$ は次の展開式で定義される．実際にこの式を使って関数の値を求める場合，n=20程度で打ち切っても十分な精度が得られることが知られている．

$$I_0(x) = 1 + \sum_{n=1}^{\infty}\left(\frac{(x/2)^n}{n!}\right)^2$$

注15：2L＋1がフィルタの係数の個数に対応し，2Lがフィルタの次数に対応する．

注16：δ と A の間には次の関係がある．$A = -20\log_{10}\delta$

Information ――可搬型外付けモバイル・ハードディスク対応ノンリニア・ビデオ編集ソフトウェア
カノープス(株)は，日本放送協会(NHK)と共同で「可搬型外付けモバイル・ハードディスク対応ソフトウェア」を開発した．このソフトウェアを用いたビデオ編集システムは，可搬型外付けモバイル・ハードディスクに構築した編集データを直接編集作業に用いることが可能になる．

す図13に示される値である．つまり，通過域および阻止域におけるリプルの最大値を δ 注16とすると，遷移域の幅 $\Delta\omega$ は，入出力の振幅比が $1-\delta$ になる角周波数 ω_P と，δ になる角周波数 ω_R との差である．L と $\Delta\omega$ の間には，標本化角周波数を ω_S とすると，次の関係(3)がある．

$$L=\frac{A-7.95}{38.72(\Delta\omega/\omega_S)} \quad \cdots\cdots(26)$$

この式から，遷移域の幅を狭くしたい場合は，フィルタの次数を高くしなければならないことがわかる．

なお，この設計方法では，最初に与える遮断角周波数 ω_C は，入出力の振幅比が0.5(= −6.02dB)になる角周波数に対応する．

● 周波数変換

この項では，最初に低域通過フィルタを設計するということで話を進めてきたが，ほかのフィルタの係数も同様の手続きにより求めることができる．しかし，周波数変換という操作を使えば，低域通過フィルタの設計結果を使ってほかのフィルタの係数を求めることができる．

元になる低域通過フィルタの係数を $h_m^{(LP)}$，その遮断角周波数を $\omega_C^{(LP)}$ とすると，ほかのフィルタの係数は表4のようになる．なお，帯域通過フィルタおよび帯域除去フィルタのところで，ω_1 は低域側の遮断角周波数，ω_2 は高域側の遮断角周波数を表す．

この表を使って，たとえば高域通過フィルタを求めるのであれば，手順は以下のようになる．設計しようとする高域通過フィルタの遮断角周波数を $\omega_C^{(HP)}$ とすると，まず $\omega_C^{(LP)}=\omega_S/2-\omega_C^{(HP)}$ を遮断角周波数にもつ低域通過フィルタの係数 $h_m^{(LP)}$ を求める．そうすると，高域通過フィルタの係数 $h_m^{(HP)}$ は，この係数 $h_m^{(LP)}$ から，$h_m^{(HP)}=(-1)^m h_m^{(LP)}$ という変換によって求めることができる．

● リアルタイム処理に対応するフィルタの係数

以上で説明したフィルタの係数は $m=0$ を中心に対称になっているので，このままではリアルタイム処理を行うフィルタに対応しない．そこで，図14に示すように，L だけ右にシフトしたものをフィルタの係数として使用することになる．

5 Parks-McClellan 法による FIR フィルタの設計法(4)

この方法は，重み付きチェビシェフ近似を使って設計する方法である．ここでも低域通過フィルタを設計するものとして話を進める．設計するフィルタの振幅特性を $|H(\omega)|$ とし，次のような仕様のものを設計することを考える．

$$|H(\omega)|=\begin{cases}1\ (\text{誤差を}\pm\delta_1\text{内とする}),\ 0\le\omega\le\omega_P\\ 0\ (\text{誤差を}\pm\delta_2\text{内とする}),\ \omega_R\le\omega\le\omega_S/2\end{cases} \quad \cdots\cdots(27)$$

設計目標であるフィルタの振幅特性を $|D(\omega)|$，重み関数を $W(\omega)$ とすると，重み付きチェビシェフ近似とは，次の式で示される誤差の絶対値 $|E(\omega)|$ の最大値を最小にする注17ことで得られるような設計法である．

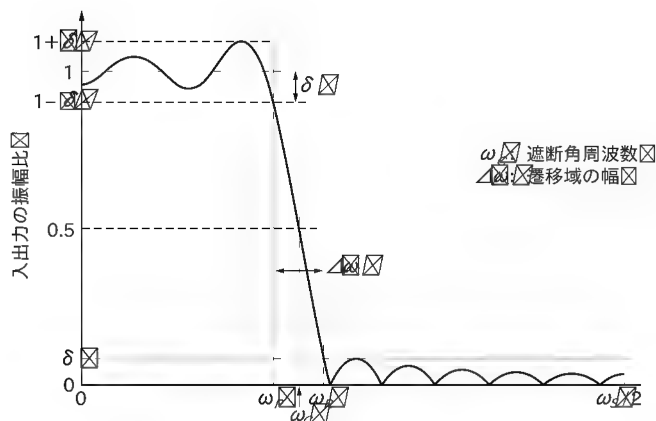
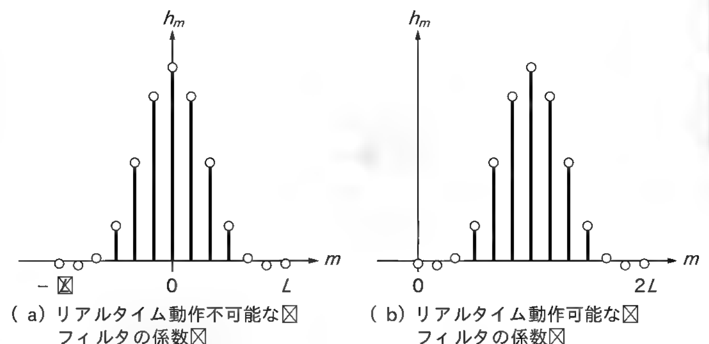


図13 カイザー窓による設計法で得られるフィルタの振幅特性



(a) リアルタイム動作不可能なフィルタの係数　(b) リアルタイム動作可能なフィルタの係数

図14 リアルタイム処理に対応する係数の変換

表4 周波数変換

	元になる低域通過フィルタの遮断角周波数	低域通過フィルタの係数をほかのタイプの係数に変換する式
高域通過フィルタ	$\omega_C^{(LP)}=\omega_S/2-\omega_C^{(HP)}$	$h_m^{(HP)}=(-1)^m h_m^{(LP)}$
帯域通過フィルタ	$\omega_C^{(LP)}=(\omega_2-\omega_1)/2$	$h_m^{(BP)}=2h_m^{(LP)}\cos n\omega_0 T$, $\omega_0=(\omega_1+\omega_2)/2$
帯域除去フィルタ	$\omega_C^{(LP)}=(\omega_2-\omega_1)/2$	$h_m^{(BR)}=1-2h_m^{(LP)}\cos n\omega_0 T$, $m=0$; $h_m^{(BR)}=-2h_m^{(LP)}\cos n\omega_0 T$, $m\neq 0$; $\omega_0=(\omega_1+\omega_2)/2$

注17：誤差の最大値が最小になるように近似を行うという手法は，数値計算の世界で，ある関数の近似式を求める場合によく使われる方法である．この手法は，最良近似またはミニマックス近似と呼ばれている(5)．

New Products ──レッドハットの組み込みソフトウェア開発用GNUProツール・スイートの最新版
レッドハットは，組み込みシステム用GNUクロス開発ツールおよびライブラリのフルスイート「Red Hat GNUPro 04r1」をリリースした．新たに，PowerQUICC III(Book E)命令セット，NEC V850アーキテクチャ，松下MN10300およびAM33コア・ファミリをサポートしたほか，最適化とユーザビリティの新しい拡張機能も多く含まれている．

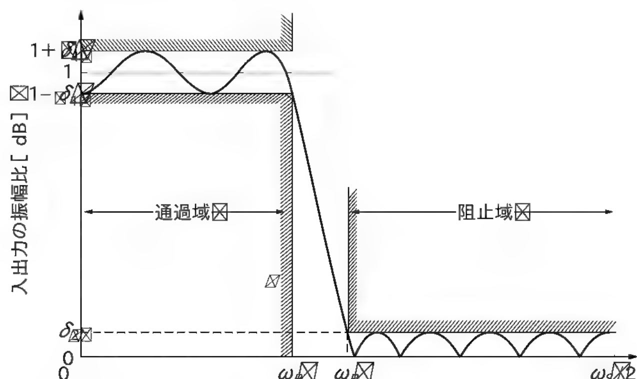


図15　重み付きチェビシェフ近似

$$|E(\omega)| = W(\omega)(|D(\omega)| - |H(\omega)|) \quad \cdots\cdots\cdots\cdots (28)$$

このとき，通常は重み関数$W(\omega)$を次のように選ぶ．通過域では$W(\omega)=W_1$という一定値，阻止域では$W(\omega)=W_2$という一定値とし，さらに$W_1\delta_1=W_2\delta_2$になるように選ぶ．この方法ではδ_1とδ_2の比を指定することはできるが，δ_1やδ_2自身を指定することはできない．

図15には重み付きチェビシェフ近似により設計される低域通過フィルタの振幅特性の一例を示す．重み付きチェビシェフ近似で設計されたフィルタの振幅特性は，通過域のリプルがすべてδ_1という同じ大きさになり，また阻止域のリプルがすべてδ_2という同じ大きさになる．このような特性は等リプル特性と呼ばれている．

このような問題を解く方法の一つとしてRemezのアルゴリズム[4]が知られている．この方法を説明するとかなり複雑な式が出てくるため，今回は説明を省略する．なお，Remezのアルゴリズムは，パラメータを少しずつ変化させながら反復的に解く方法なので，場合によっては解が求められない場合もある．

6 FIRフィルタの設計プログラム

前々項および前項で説明した方法でプログラムを作るのは，かなりたいへんな仕事になる．そこで，筆者の作成したプログラムを紹介する．このプログラムはBorland社のC++ Builder 6.0で作成したもので，実行ファイルやソース・ファイル，このプログラムの中で利用しているインクルード・ファイルやライブラリなどのうち，筆者の作成したものは本誌のWebサイトからダウンロードできる．また，InterGiga No.34に収録する予定である．

なお，これらのプログラムは，設計した係数をファイルに保存できるように作成した．このファイルに保存された係数は，次項で説明するプログラムで利用できるようになっている．

● カイザー窓によるフィルタ設計のプログラム

カイザー窓によるフィルタ設計プログラム(`FIR_Kaiser.`

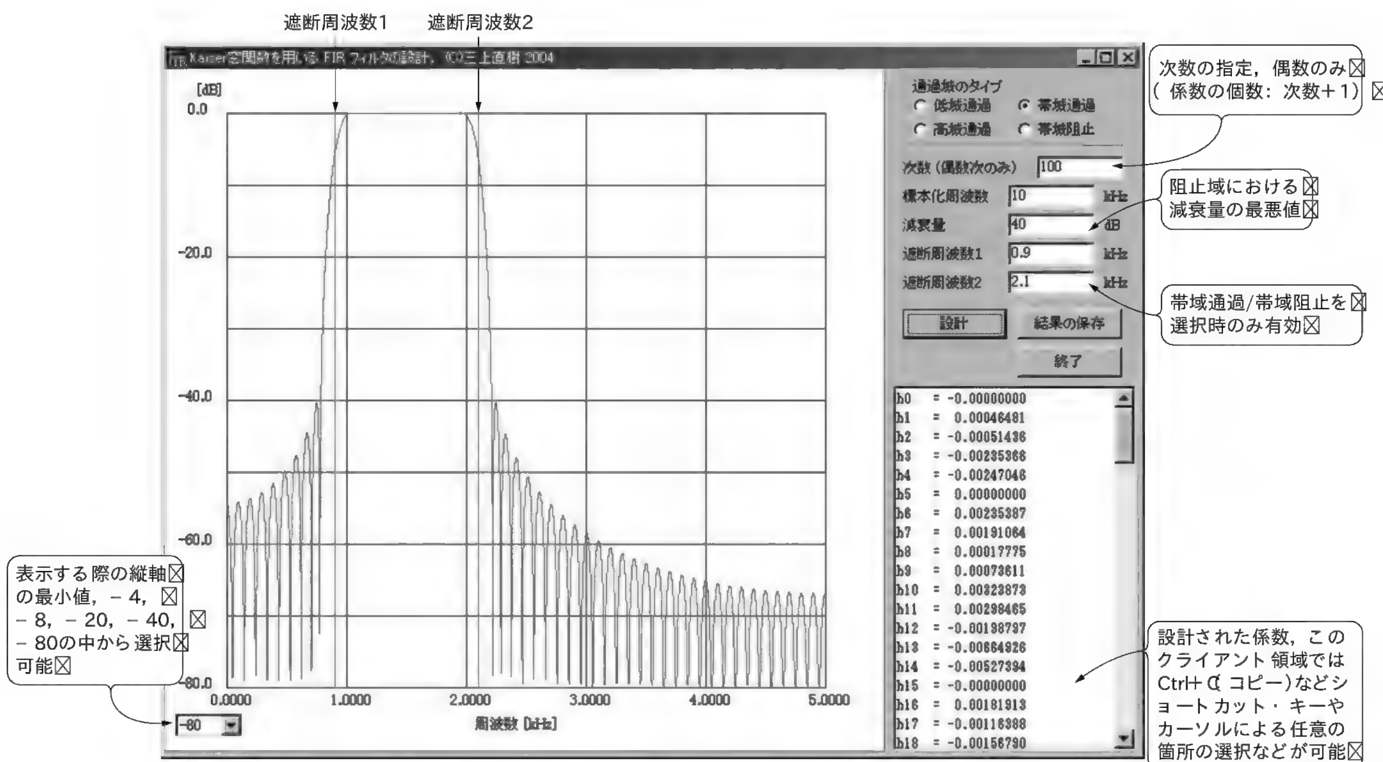


図16　カイザー窓を用いるFIRフィルタ設計プログラムの実行例
帯域通過フィルタの設計の場合

New Products——医療用超音波システムに高画質を実現する低消費電力クワッド可変ゲイン・アンプ
アナログ・デバイセズは，低消費電力化への要求が厳しい医療用超音波システムなど向けに，最適化された低コストのクワッド可変ゲイン・アンプ(VGA)「AD8335」を発売した．5V単一電源で動作し，消費電力はチャネルあたり92mW．

exe)で，帯域通過フィルタを設計したときのようすを図16に示す．この例では，次数= 100次，標本化周波数= 10kHz，減衰量の最悪値= 40dB，遮断周波数1= 0.9kHz，遮断周波数2= 2.1kHzという設計パラメータを与えた場合である．この二つの遮断周波数も，設計されたフィルタの振幅特性と併せて表示される．PCの画面の上では，この遮断周波数はマゼンタ色で表示される．

設計の際は，通過域のタイプ，つまり低域通過，高域通過，帯域通過，帯域阻止の四つの中から選択することができる．次数[注18]は偶数次に限定している．そのほか，標本化周波数，減衰量の最悪値(図12の*A*)，遮断周波数を指定する．帯域通過/帯域阻止フィルタの場合は低域側の遮断周波数(遮断周波数1)と高域側の遮断周波数(遮断周波数2)を指定する．以上の指定を行った後に[設計]ボタンをクリックすると係数が設計され，それに対応する振幅特性と与えた遮断周波数が表示される．通過域を拡大して表示したい場合は，左下のドロップダウン・リスト・ボックスの数値を選択する．

● Parks-McClellan法によるフィルタ設計のプログラム

Parks-McClellan法によるフィルタ設計プログラム(`FIR_Remez.exe`)で，帯域通過フィルタを設計したときのようすを図17に示す．この例では，次数= 100次，標本化周波数= 10 kHz，そのほかのパラメータは図17に示す値を与えた場合である．

次に，パラメータを与える場合に注意する項目について説明する．

<帯域数の合計>

低域通過/高域通過フィルタを設計する場合は"帯域数の合計"の項目を2とする．帯域通過/帯域阻止フィルタを設計する場合は，この項目を3に設定する．この値は5まで指定できるので，通過域や阻止域の合計が5の場合まで対応できる．

<利得>

通過域に対応する帯域では1，阻止域に対応する帯域では0を指定する．この項目は正の値であれば0や1以外を指定してもよい．

<重み>

各帯域のリプルの大きさに反比例する値を指定する．つまり，

注18：次数+ 1が係数の個数である．

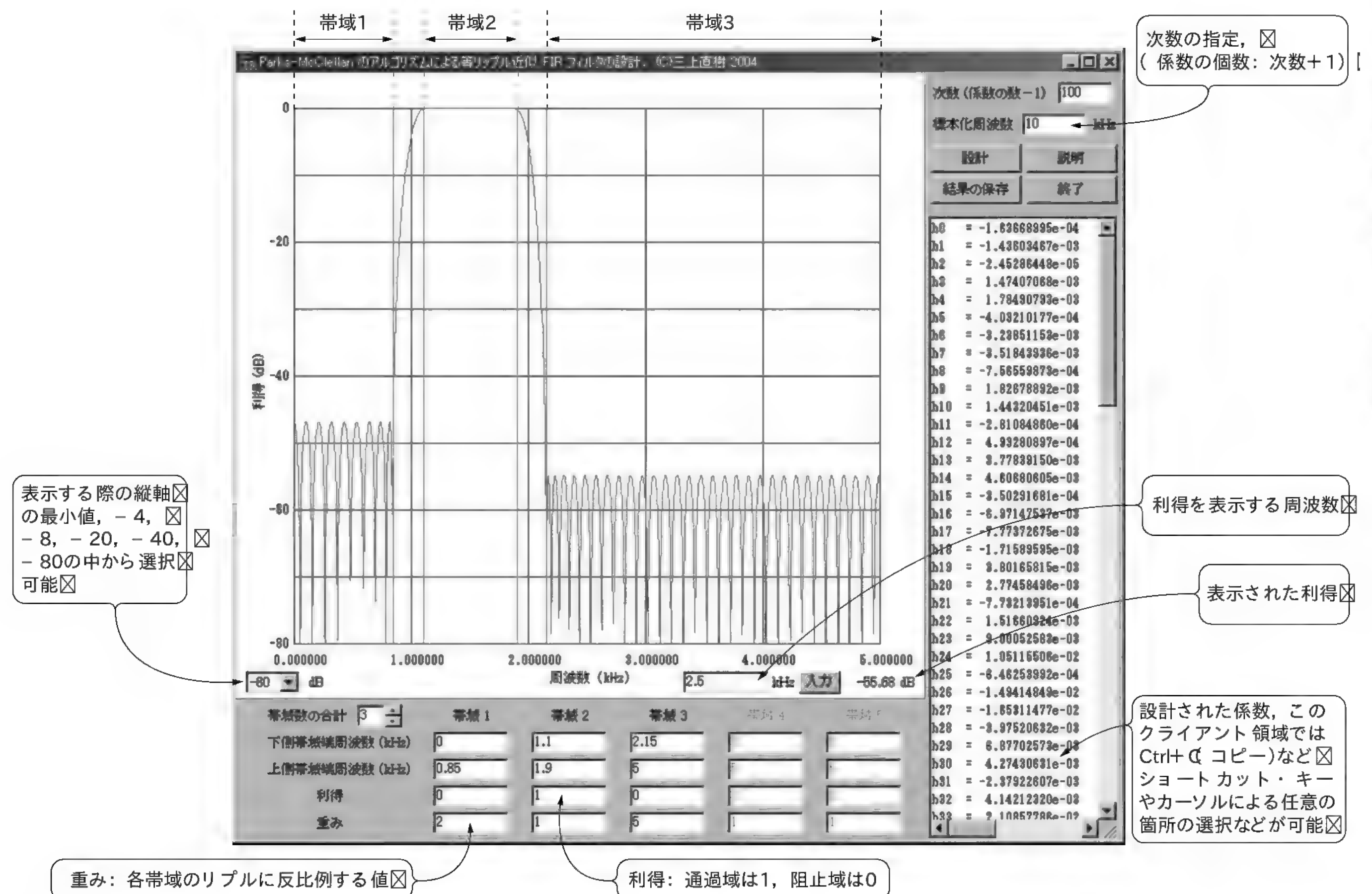


図17　Parks-McClleran法によるFIRフィルタ設計プログラムの実行例
帯域通過フィルタの設計の場合

New Products —— TIのモバイル・アプリケーションを革新する低消費電力DSP新シリーズ3種と設計ツール
日本テキサス・インスツルメンツ(株)は，低消費電力DSP(ディジタル・シグナル・プロセッサ)「TMS320C5509A」，「同C5507」，「同C5503」の3製品および低消費電力設計ツール「eXpressDSPパワー設計ツール」を発表した．スタンバイ電力0.12mW，機能ユニットを個別にON/OFFできることが特徴．



図 18　係数の表示領域を拡大したようす



図 19　Parks-McClellan 法で解が収束しない場合のメッセージ

帯域 j における重みを W_j，リプルの大きさを δ_j とすると，次の式を考慮して各帯域の重みを指定する．

$$W_1 : W_2 : W_3 : \cdots \boxtimes \frac{1}{\delta_1} : \frac{1}{\delta_2} : \frac{1}{\delta_3} : \cdots \quad (29)$$

なお，Parks-McClellan 法ではリプルの大きさである δ_j そのものを指定することはできない．

設計された係数を表示する領域には，設計されたフィルタの特性を表す数値も表示されている．その部分を見るためには，係数が表示されている領域のスクロール・バーを下まで持っていき，このアプリケーションが表示されているウィンドウの右側をマウスでドラッグして表示の幅を拡大すればよい．拡大したときの表示の中で，該当する箇所を抜き出して，図 18 に示す．これを見ると，各帯域におけるリプルの大きさの比が 1/2：1/1：1/5 になっているが，これは設計する際に三つの帯域の重みをそれぞれ 2，1，5 にしたことによる．

このプログラムでは，設計されたフィルタの振幅特性について，周波数を指定するとそのときの利得（入出力の振幅比）を求めることができるように作成した．図 17 において矢印で"利得を表示する周波数"と示した箇所のテキスト・ボックスに周波数を入力し，[入力]ボタンをクリックすると，その右側にその周波数の利得が表示される．この例では 2.5kHz を指定し，そのときの利得が－55.68dB と表示されている．この周波数はマウスのカーソルを振幅特性が表示されている領域に持っていくことでも指定することができる．

左下のドロップダウン・リスト・ボックスは，カイザー窓によるフィルタ設計プログラムの場合と同様に，表示の最小値を選択できる．

注 19：周波数 0 と標本化周波数の 1/2 になる場合は除く．
注 20：関数 `FIR()` を呼び出す際に，第 1 引き数が `n` ではなく，`n+order` になっている．

リスト 1　FIR フィルタのデモ・プログラミングの，FIR フィルタ実行に関する部分

```
// フィルタ処理の実行
    for (int n=0; n<nData-MaxORDER; n++)
        yn[n] = FIR(n+order, xn, hm, order);



// FIR フィルタの実行
double FIR(int nIN, double xn[], double hm[], int nOrder)
{
    double output = 0.0;
    for (int n=0; n<=nOrder; n++)
        output = output + xn[nIN-n]*hm[n];
    return output;
}
```

各帯域で指定した下側遮断周波数と上側遮断周波数は，設計されたフィルタの振幅特性と併せて表示される[注19]．PC の画面の上では，この遮断周波数はマゼンタ色で表示される．

Parks-McClellan 法は，反復的な手法で解を求めるので，設計パラメータの与え方によっては収束せずに解が求められない場合がある．その場合は図 19 に示すメッセージを出して停止するので，そのときはパラメータを変えて再度実行する必要がある．

7 Windows 上で実行可能な FIR フィルタのデモ・プログラム

FIR フィルタの動作を視覚的に見ることができるプログラムを作成した．このプログラムでは，フィルタの係数は，前項で示した設計プログラムで得られた係数のファイルから読み込むようになっている．

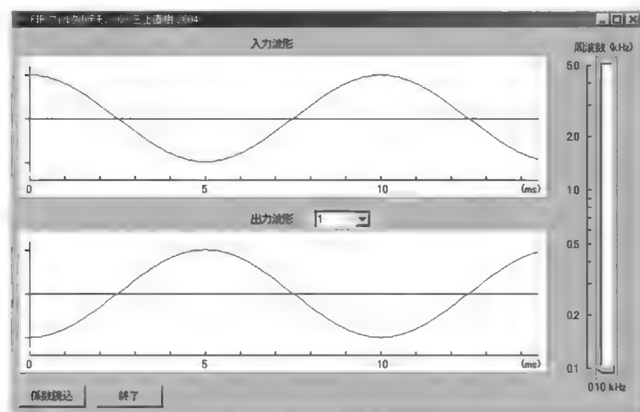
プログラムの中で，FIR フィルタの実行に関する部分を抜き出してリスト 1 に示す．プログラムの全体は本誌の Web ページからダウンロードできる．また，InterGiga No.34 に収録する予定である．

関数 `FIR()` は，`nIN` の時点における FIR フィルタ処理に対応する．入力信号は第 2 引き数の `xn` に，フィルタの係数は第 3 引き数の `hm` に与える．第 4 引き数 `nOrder` にはフィルタの次数を与える．この関数は入力信号 `xn[nIN]`，`xn[nIN-1]`，`xn[nIN-2]`，…，`xn[nIN-nOrder]` から，`nIN` の時点における出力を計算するように作られている．

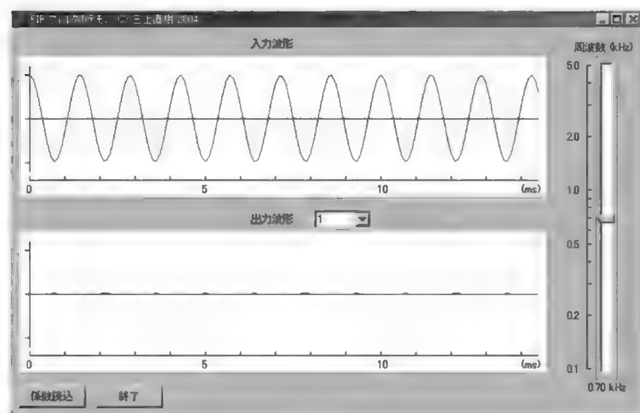
この関数 `FIR()` を `for` ループで順に実行していけば FIR フィルタが実行される．なお，このとき開始点が 0 ではなく `order`（フィルタの次数）になっている[注20]のは，フィルタの計算の際に，現在の時点の入力信号から `order` だけ過去の信号を使って，現在の出力を計算するからである．

このプログラムでは標本化周波数が 10kHz であると想定して，表示の際の時間軸の目盛りを付けている．そのため，このプログラムで使う係数を設計する際には，標本化周波数を 10kHz に設定して設計するほうがよい．

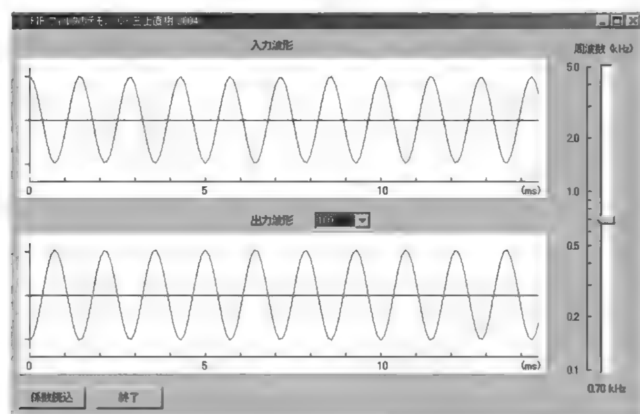
New Products ——村田製作所，2.0GHz 帯 W-CDMA 端末用小型・軽量の表面波デュプレクサを発売
（株）村田製作所は，2.0GHz 帯 W-CDMA 端末対応の小型・軽量の表面波デュプレクサ「SAYCS1G95AA0N05」を発売した．3.2×5.0×1.2mm（max.）のセラミック・パッケージを採用し，従来の誘電体デュプレクサに比べ面積比 41 %の小型サイズを実現した．



(a)入力信号周波数= 100Hz の場合



(b)入力信号周波数= 700Hz の場合



(c)入力信号周波数= 700Hz で，出力の振幅を 100 倍にした場合

図 20　FIRフィルタのデモ・プログラムを実行したようす

表 5　設計の際に与えたパラメータ

次　数	100次	
標本化周波数	10 kHz	
	帯域1	帯域2
下側帯域端周波数	0	0.7
上側帯域端周波数	0.5	5
利　得	1	0
重　み	1	1

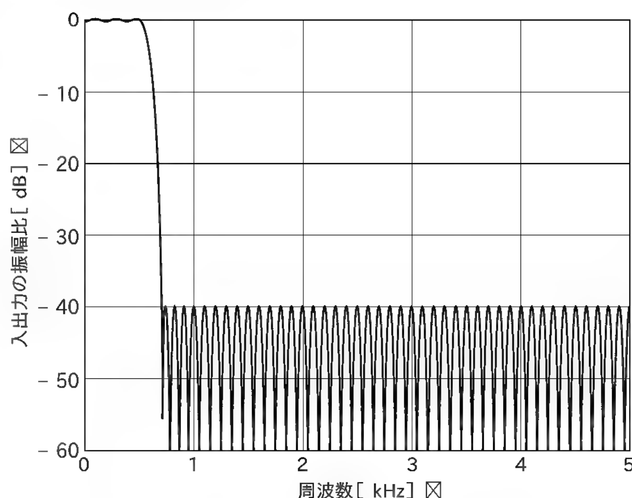


図 21　図 20に示す FIRフィルタのデモ・プログラムで使用したフィルタの振幅特性

このプログラムを実行したときのようすを図 20に示す．[係数読込]ボタンをクリックし，前項の設計プログラムで得られた係数ファイルを読み込むと，フィルタの出力波形が現れる．画面右側のスライダを動かすと，入力信号の周波数が変化し，それに対応して入力信号と出力信号の波形が更新される．

この例では，Parks-McClellan 法を使い，表 5のパラメータを与えて設計したフィルタの係数を使った．このときのフィルタの振幅特性は図 21のようになる．

図 20(a)は，入力信号の周波数が 100Hz の場合で，出力には同じ振幅の波形が現れている．図 20(b)は入力信号の周波数が 700Hz の場合で，この周波数は阻止域に当たるので，出力波形の振幅はほとんど 0である．このプログラムは，出力波形の振幅を拡大できるようになっている．画面の"出力波形"の表示の右側にあるドロップダウン・リスト・ボックスの数値を選択することで拡大できる．そこで，図 20(b)の状態で，100倍に拡大したものが図 20(c)である．出力波形を 100倍すると入力波形の振幅とほぼ同じなので，出力の振幅が入力の振幅の約 1/100になっていることがわかる．

8 DSPで実現するFIRフィルタ

最後に，FIRフィルタを DSP 上で実行するためのプログラムを示す．この場合に，前項で示したプログラムと大きく異なる点は，リアルタイム動作を考慮してプログラムを作る必要があるということである．また，DSP が実際の製品に組み込まれた形で使われる場合に，演算は固定小数点演算で行われる場合が多いので，ここでは固定小数点演算を使ったプログラムを示す．

● FIRフィルタの構成

FIRフィルタを実現するために行われる計算は，基本的に式(6)である．これをブロック図で表すと，図 22のようになる．

New Products ――アルプス電気，Bluetooth の All in one モジュールと小型・軽量 HCI モジュールを発売
アルプス電気(株)は，Bluetooth の新モジュール 2シリーズを発売した．プロトコル・スタック，プロファイルを盛り込んだ All in one タイプ・モジュール「UGZZ2シリーズ」と容積 0.075mlの小型・高密度化を実現した低消費電流対応タイプ HCI モジュール「UGNZ2シリーズ」である．

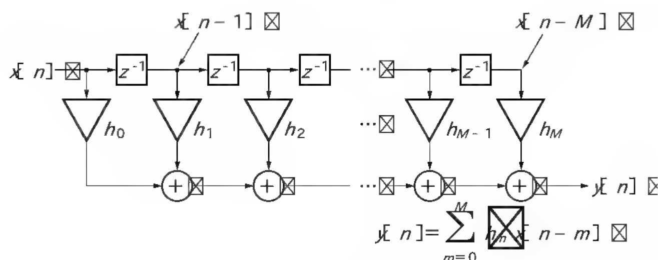


図22　直接型FIRフィルタのブロック図

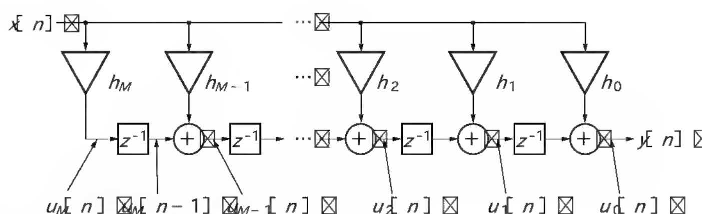


図23　転置型FIRフィルタのブロック図

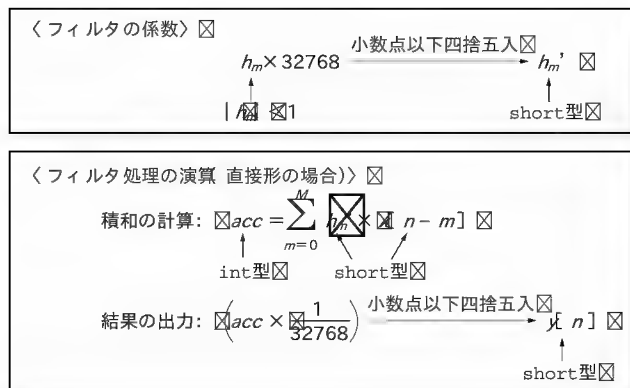


図24　固定小数点演算による直接型FIRフィルタ処理のようす

このような構成法は直接形(direct form)と呼ばれている．このほかにも，式(6)の処理と等価な処理を行うことができるいくつかの構成法があり，直接形の転置形[以下では単に転置形(transposed form)と呼ぶ]がその一例である．

転置(transpose)とは，ブロック図において以下の三つの操作を行うことである．

1) 入力と出力を交換する
2) 信号の流れをすべて逆転する
3) 加算器と分岐点を交換する

このような操作によって得られる転置形のブロック図を図23に示す．このブロック図に対応する差分方程式は，図23の中に示す $u_m[n]$，m=0，1，…，M を使って，次のように表すことができる．

$$\begin{cases} u_M[n]=h_Mx[n] \\ u_{M-1}[n]=h_{M-1}x[n]+u_M[n-1] \\ \vdots \\ u_1[n]=h_1x[n]+u_2[n-1] \\ u_0[n]=h_0x[n]+u_1[n-1] \\ y[n]=u_0[n] \end{cases} \quad \cdots\cdots (30)$$

● FIRフィルタのプログラム

Texas Instruments社(TI社)のTMS320C6713 DSPスターター・キットの上で動くFIRフィルタのプログラムをリスト2(`FIR1.cpp`)に示す．このプログラムやその中で使われているインクルード・ファイルなどのプロジェクト一式は本誌のWebサイトからダウンロードできる．また，InterGiga No.34に収録する予定である．

直接形には関数`Direct()`，転置形には関数`Transposed()`が対応している．フィルタの係数はParks-McClellan法で求めたもので，帯域通過フィルタになっている．設計の際に与えたパラメータなどはリスト2の中のコメントを参照のこと．

プログラムでは最初に，関数`Direct()`で作業領域として使用する配列`xn`と，関数`Transposed()`で作業領域として使用する配列`un`に0を代入して初期化を行う．次に，`while`によるループ処理に入る．関数`Input()`，`Output()`は入力および出力の関数で，2チャネルに対応しており，データは`short`型(16ビット)になっている．この二つの関数はインクルード・ファイル`AIC23_IO.hpp`の中で定義されている．関数`Input()`は，設定された標本化間隔ごと(このプログラムでは1/48,000 s)にデータを取得する．チャネル0の信号は関数`Direct()`で，チャネル1の信号は関数`Transposed()`で処理を行う．

<直接形，**`Direct()`**>

固定小数点演算でFIRフィルタを実現する場合にまず考えなければならないことは，データの表現方法である．これを図24を使って説明する．

このプログラムでは係数を`short`型(16ビット)で表している．一方，FIRフィルタの係数 h_m は一般に，その絶対値が1よりも小さくなる．そこで，図24の上の図で示すように，設計プログラムで求められた係数 h_m に32,768(=2^{15})を乗算し，小数点以下を四捨五入したものを使う．これを h_m' とする．入力信号 $x[n]$ は16ビットなので，そのまま`short`型のデータとする．

フィルタの処理に対応する計算では，図24の下の図で示すように，入力信号 $x[n]$ と係数 h_m' との積和計算を行い，それを acc へいったん格納する．入力信号と係数はどちらも16ビットなので，積和の結果を正確に表すためには32ビット必要になる．したがって，プログラムで積和の結果が格納される変数 acc は32ビット必要になるため，`int`型として宣言されている．

以上の結果，図24に示すように acc へは本来の値を32,768倍したものが格納される．したがって，これを32,768で割り算を行い，小数点以下を四捨五入したものが最終的な結果となる．この割り算は`int`型データの15ビット右シフトで実現できるが，そのままでは小数点以下を四捨五入したことにはならない

New Products――アンカーシステムズ，FreeLines H8/3048F評価ボード用Ethernetボードを発売
アンカーシステムズ(株)は，(有)フリーラインズ製のH8/3048F評価ボード「FreedomH8評価ボード(FLDM-H8CPU-MB)」用に，オプションのEthernetボードを発売した．Freedomシリーズは，USBやPC/104バスなどを搭載し，TOPPERS/JSPカーネルやコンパイラ，デバイス・ドライバが付属している．

リスト 2　DSP で実現した FIR フィルタのプログラム(FIR1.cpp)

```
//----------------------------------------------------------------------
// FIRフィルタ (帯域通過フィルタ)
// 次数        100
// 標本化周波数  48 kHz
//                       帯域1          帯域2          帯域3
// 下側帯域端周波数 (kHz)  0.00000000     2.00000000     5.00000000
// 上側帯域端周波数 (kHz)  1.00000000     4.00000000    24.00000000
// 利得                  0.00000000     1.00000000     0.00000000
// 重み                  1.00000000     1.00000000     1.00000000
// リプル                0.01186177     0.01186177     0.01186177
// リプル (dB)         -38.51701198     0.10242373   -38.51701198
//----------------------------------------------------------------------
#include "AIC23_IO.hpp"    // 信号の入出力用

inline short Round15(int x) { return (x + 0x4000)>>15; }

short Direct(const short x, const int M, const short hk[], short xk[]);
short Transposed(const short x, const int M, const short hk[], int uk[]);

const int ORDER = 100;
const short hn[ORDER+1] =
{
       93,    -131,     -20,      36,      53,      39,       0,     -57,    -122,    -178,
     -209,    -202,    -157,     -82,       0,      62,      86,      62,       1,      69,
     -112,     -95,       0,     162,     359,     535,     634,     617,     478,     251,
        1,    -191,    -263,    -189,       0,     221,     363,     313,       1,    -563,
    -1290,   -2010,   -2520,   -2632,   -2230,   -1311,      -1,    1471,    2817,    3759,
     4097,    3759,    2817,    1471,      -1,   -1311,   -2230,   -2632,   -2520,   -2010,
    -1290,    -563,       1,     313,     363,     221,       0,    -189,    -263,    -191,
        1,     251,     478,     617,     634,     535,     359,     162,       0,     -95,
     -112,     -69,       1,      62,      86,      62,       0,     -82,    -157,    -202,
     -209,    -178,    -122,     -57,       0,      39,      53,      36,     -20,    -131,
       93
};

short xn[ORDER+1];// 直接形のためのバッファ
int   un[ORDER+2];// 転置形のためのバッファ

int main()
{
    short ch0_in, ch1_in, ch0_out, ch1_out;

    for (int k=0; k<=ORDER; k++) xn[k] = 0;
                            // 直接形のためのバッファをクリア
    for (int k=0; k<=ORDER; k++) un[k] = 0;
                            // 転置形のためのバッファをクリア

    while(1)
    {
        Input(ch0_in, ch1_in);        // AD 変換器からの入力

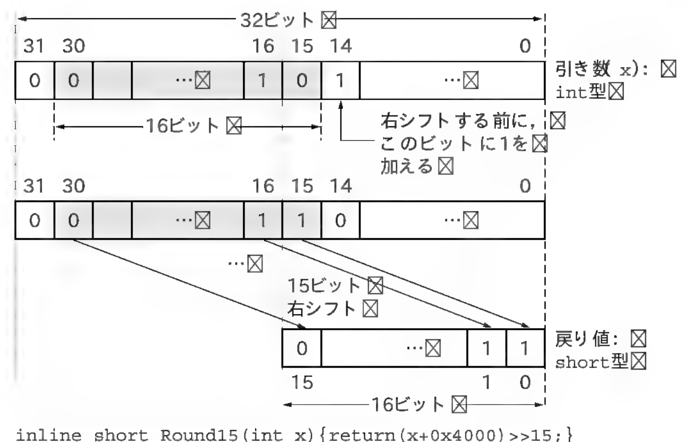
        ch0_out = Direct(ch0_in, ORDER, hn, xn);
                                  // 直接形 FIR フィルタの実行
        ch1_out = Transposed(ch1_in, ORDER, hn, un);
                                  // 転置形 FIR フィルタの実行
        Output(ch0_out, ch1_out);     // DA 変換器への出力
    }
}

// 直接形 FIR フィルタ
short Direct(const short x, const int M, const short hm[],
                                                     short xm[])
{
    xm[0] = x;
    int acc = 0;
    for (int m=0; m<=M; m++) acc = acc + hm[m]*xm[m];
                                                   // 積和の計算
    for (int m=M; m>0; m--) xm[m] = xm[m-1];       // データの移動
    return Round15(acc);
}

// 転置形 FIR フィルタ
short Transposed(const short x, const int M, const short hm[],
                                                   int um[])
{
    for (int m=0; m<=M; m++) um[m] = hm[m]*x + um[m+1];
    return Round15(um[0]);
}
```

ので，右シフトを行う前に，第14ビット目に1を加えてからシフトを行うようにする．この処理を実現するのがインライン関数の Round15() で，その処理のようすを図25に示す．

直接形FIRフィルタを実現する場合は図22のブロック図に従ってプログラムを書くことになる．入力信号は標本化間隔ごとに得られるので，新しい信号が入力されるたびに，図22の遅延素子に格納されているデータを一つずつ右側へ移動しなければならない．これをプログラムで実現する場合，データ自身を移動する方法と，データを指すポインタを移動するという2通りの方法が考えられる．ここではデータ自身を移動するという方法を採用することにする．関数 Direct() の中では，遅延器には配列 xm が対応する．このときのデータの移動のようすを図26に示す．



New Products ── アナログ・デバイセズ，シングルチップ高速デュアル 14 ビット ADC ファミリを発売
アナログ・デバイセズ社は，2個の14ビット A-Dコンバータを1チップに集積した業界初の14ビット・デュアルADC「AD92xx」ファミリを発売した．9×9mmのチップ・スケール・パッケージ(CSP)を採用し，20～105Msps(サンプル/秒)のスピードで10ビット，12ビット，14ビットの分解能を提供する．

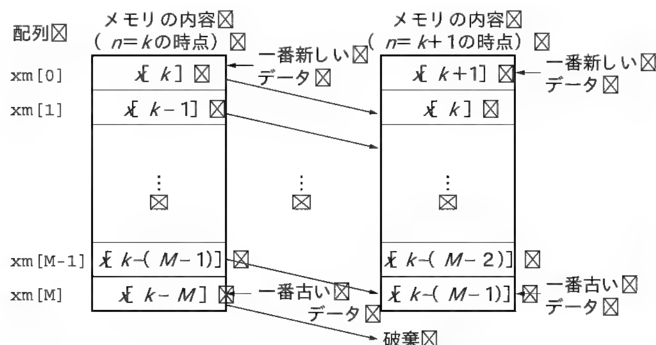


図26　直接型FIRフィルタ実行時の，入力信号の配置と移動のようす
*M*次のフィルタの場合．固定小数点演算による転置型FIRフィルタ処理のようす

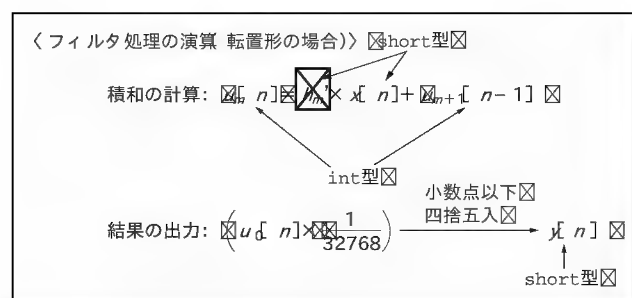


図27　固定小数点演算による転置型FIRフィルタ処理のようす

以上のことから，直接形を実現する場合は，リスト2に示すように，主要な処理は二つの`for`ループから構成されるプログラムになる．

<転置形，**`Transposed()`**>

転置形の場合も，係数については直接形と同じように，32,768倍した値を使う．処理のようすを図27に示す．最終的な結果は直接形と同様に，32,768で割り算を行い，小数点以下を四捨五入したものになる．

転置形の場合，`for`ループは一つだけのプログラムになる．

● 実行結果

写真1には入力に1kHzの矩形波を入力した場合の結果を示す．上の波形が入力信号，下の波形が出力信号である．矩形波は基本周波数とその奇数倍の周波数成分をもつので，この場合1.0，3.0，5.0，7.0，…kHzという周波数成分をもつことになる．ここで作成したFIRフィルタは，この周波数成分の中で3.0kHzの成分のみを通す．したがって，フィルタの出力には，写真1の下に表示されている波形のように，周期が入力の矩形波の1/3の正弦波，つまり3kHzの正弦波が出力として得られることになる．

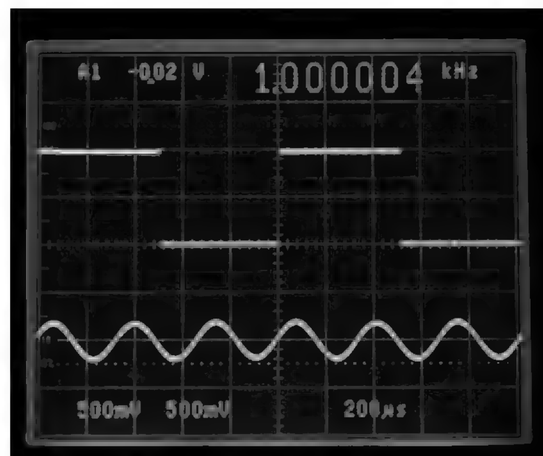


写真1　1kHzの矩形波を入力した場合

なお，このプログラムは片方のチャネルは直接形で，もう一方は転置形でフィルタ処理を行っているが，同じ入力信号に対しては，まったく同じ出力信号が得られる．

参考文献
(1) 気象庁のWebサイト，電子閲覧室，http://www.data.kishou.go.jp/より引用．
(2) 武部 幹；ディジタルフィルタの設計，2.5節，東海大学出版会，1986年．
(3) R. W. Hamming著，宮川，今井訳；ディジタル・フィルタ，第9章，科学技術出版社，1980年．
(4) L. R. Rabiner, J. H. McClellan, T. W. Parks, "*FIR digital filter design techniques using weighted Chebyshev approximation,*" Proceedings of IEEE, vol. 63, pp.595-610, 1975.
(5) 一松 信；初等関数の数値計算，教育出版，1974年．

みかみ・なおき　職業能力開発総合大学校 情報工学科

New Products —— 日本語配列・かな無刻印モデル Happy Hacking Keyboard Lite2
(株)PFUは，コンパクト・キーボード Happy Hacking Keyboard Lite2に，日本語配列・かな無刻印モデルを追加した．同キーボードはAの左にCTRLキーが存在するなど，UNIX環境での使用にとくに適する．オープン価格．

3 ディジタル・フィルタを体感
DSPで実現する音声処理アプリケーションの開発

音声処理アプリケーションとして，DSPを用いてディレイ（エコー）やリバーブなどのエフェクタを作成する．フィルタの基礎やアルゴリズムの理解に加えて，作成したエフェクタを使って，ディジタル・フィルタをじかに体験してほしい．なお，これらのエフェクタをシミュレートするアプリケーションが，本誌のWebサイトからダウンロードできる．
（編集部）

野澤 直哉

はじめに

DSP（Digital Signal Processor）は，文字どおり「信号をディジタル数値の形式で処理するもの」である．そういう意味では，そろばんもDSPということになる．このように，ディジタル信号処理自体は決して新しい技術ではないが，今では音声処理に限らず，画像処理や通信用，計測用データ処理には欠かせないものとなっており，優秀な開発者が多く求められるようになってきた．

最近では多くのメーカからDSPの評価ボードが発売されていて，これらを教材としてDSPのためのアルゴリズム開発者は数多く育ってきている．しかしDSPのハードウェアを開発できる人はまだ少数といわざるをえないのではないだろうか．

FPGAなどのプログラマブル・デバイスを用いれば，DSPのIPを使用することができるので，DSPの機能をもったものを作ることは難しくないだろう．しかし，これらは内部アーキテクチャを公開していないものが多く，参考にはなりにくい．

特許の問題や，処理効率の点で市販のDSPでは仕様を満足できない場合があり，独自の特徴を持ったDSPが求められるケースは多い．こういった場合，ハードウェアのプリミティブな構成を理解することが非常に重要になってくる．

本章では，DSPのハードウェア開発手法を学ぶ人向けに開発したゼクー社製のDSPボード「xDSP-1」（pp.74-75）を使って解説を進めていく．具体的には，音声や楽音に効果を付けるエフェクタを例に挙げ，必要なアルゴリズムとその実現に必要なハードウェア技術を解説する．また，マイクロプロセッサ（またはマイコン）とDSPは本質的に同じものなので，マイコンの開発者にも有用であると思われる．

xDSP-1は，DSPアーキテクチャを学ぶことを前提としているので，DSPはFPGAに実装する．FPGAにはAltera社ACEXシリーズのEP1K30-3を使用する．

1 ハードウェアの開発手順

まず，音声信号処理用DSPに要求される基本仕様を考え，それを実現できる最低限のアーキテクチャを設定する．

なお，本章では，被乗数（信号系）を「データ」，乗数を「係数」と呼ぶことにする．また，乗算と，その結果を積算することをまとめて「積和」と呼ぶことにする．

1）マルチエフェクタを実現できること

まず，5～6台のエフェクタを同時に実行できる処理能力が望まれる．エフェクタ1台あたり，10～30演算を要するので，1サンプルあたり100回程度の積和を実行できればよいことになる．これを元に下記のように数値を決定する．

- 1サンプルあたりの積和数は，少し余裕を持って125回とする
- サンプリング周波数は，32（kHz）とする
- システム・クロックを32（MHz）とすると，演算時間は積和1回あたり250nsとなる

250nsに1回の積和が実行できればよいので，「シフト加算方式」で乗算を実現することにする．シフト加算とは，シフタと加算器を組み合わせて時分割で乗算するものである．メリットとしては，

- 乗算器を実装するよりもゲート数が小さい
- 消費電流が低減でき，システムのトータル・コストを下げることにもなる

といったことが挙げられる．一方，デメリットとしては，

- 1回の積和に数クロックが必要なので，演算の効率が悪い
- タイミング回路が複雑になる

などがある．しかし，最近のFPGAは高速なので，クロック周波数を上げて時分割処理を行っても十分に実用になる．

2）汎用DSPであること

特定の用途に使うのではなく，プログラムしだいでいろいろ

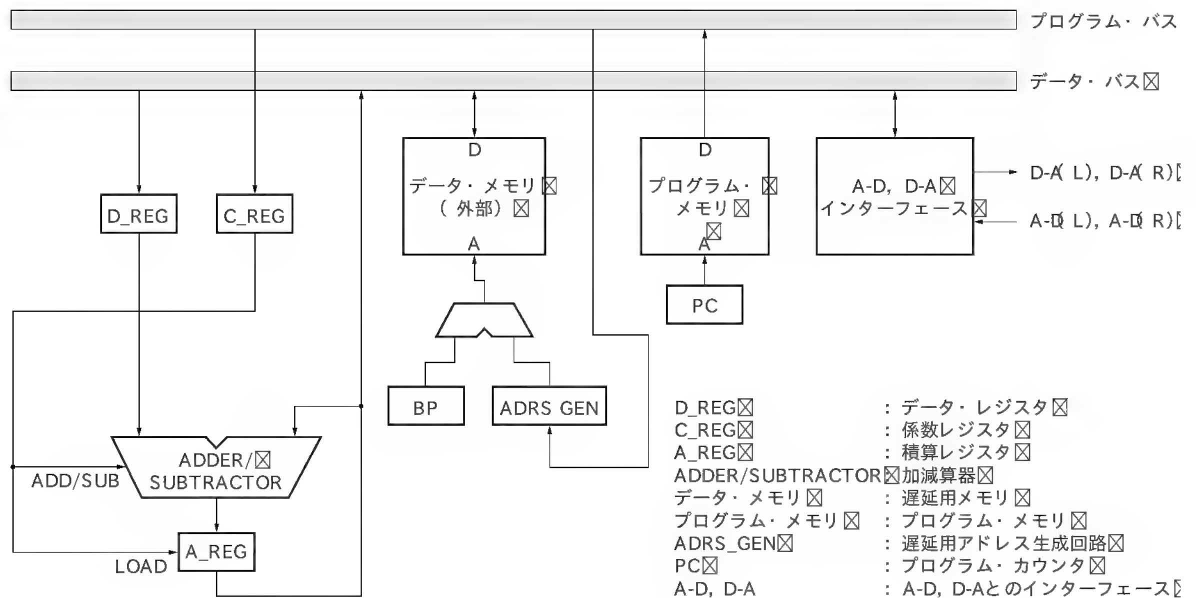


図1 基本アーキテクチャ

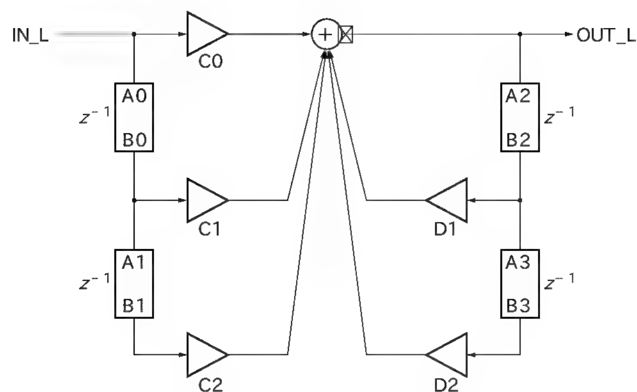


図2 Bi-Quadフィルタの構成

な機能をもつためには，プログラムRAMをもつことが必要となる．基本的には，1サンプルあたりの積和数と同じ125ワードあればよいが，条件分岐があるとプログラムに余裕が必要となるので，256ワードとする．ビット長は24とする．

3) 任意の遅延時間を設定できること

ディレイなどの遅延時間を得るために，

- 数百msのディレイ用メモリをもつ
- 外部にSRAMを実装することで実現する
- 必要な容量を16,384ワード×24ビットとする

とし，これで最大512msの遅延が実現できる．

4) ステレオ入力とステレオ出力をもつこと

CDなどと同程度のダイナミック・レンジを確保するため，16ビット精度で，2チャネルのA-D，D-Aコンバータが必要となる．

以上の四つの事項を踏まえ，図1に基本アーキテクチャを示す．

2 フィルタの設計とディレイ・ライン

この時点では，データ語長，係数語長などが未定であるので，フィルタの設計を通じて決定していくことにする．さらに，オーディオ・エフェクタに必要な機能も追加し，図1の基本アーキテクチャに肉付けしていき，最終的には章末の図B(p.75)のアーキテクチャを採用した．

オーディオ・エフェクトには，フィルタが数多く使われており，フィルタを効率よく実装することがDSP設計の鍵となる．

アナログ・フィルタの特性を近似的にディジタル・フィルタに変換して設計する手法はいろいろとあり，「$s \rightarrow z$変換」と呼ばれている．アナログ・フィルタには過去に設計された資源が豊富にあり，その設計手法も充実しているので，設計の効率化が図れる．ここでは，変換を行う手法の一つである「双一次変換」を使い，実際によく使うBi-Quadフィルタを設計していく．

Bi-Quadフィルタは，図2のようなフローで示される．この伝達関数は，式(1)のようになる．

$$\frac{OUT_L}{IN_L} = \frac{C_0 + C_1 z^{-1} + C_2 z^{-2}}{1 - D_1 z^{-1} - D_2 z^{-2}} \quad \cdots\cdots\cdots(1)$$

このフィルタの特徴としては，

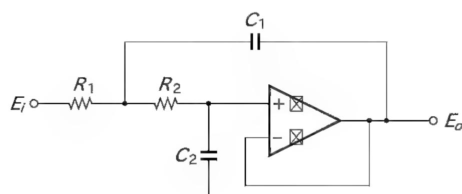
- 係数のパラメータを変えるだけで，さまざまな特性のフィルタを作ることができる
- データ語長が小さくても，ダイナミック・レンジを確保しやすい

などが挙げられ，よく使われている．

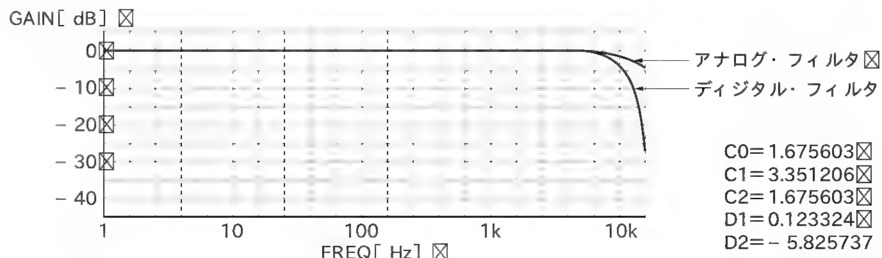
ここで使われているz^{-1}は，1サンプル分の遅延を表している．図2のようにz^{-1}を2個縦続接続すると，z^{-2}となる．

双一次変換では，アナログとディジタルの周波数領域の間に

New Products——日本テクトロニクス，信号発生器の操作性を向上させるソフトウェアを無償配布
日本テクトロニクス(株)は，同社の任意波形ジェネレータ「AWG710B型」，「AWG615型」用に波形生成・編集ツール「ArbExpress」を開発し，無償で配布を開始した．同ソフトウェアはhttp://www.tektronix.co.jp/よりダウンロードできる．



(a) アナログ・ローパス・フィルタ



(b) ローパス・フィルタの振幅と周波数特性

図3 ローパス・フィルタ回路

表1 フィルタの設計条件

サンプリング周波数	32kHz
カットオフ周波数	10kHz
通過域の利得	0dB
フィルタの型	VCVS法

次に示す関係がある．

$$s=\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}\text{，}T\text{：サンプリング周期}\quad\cdots\cdots(2)$$

双一次変換とは，アナログ・フィルタの伝達関数のsに式(2)を代入して，ディジタル・フィルタの伝達関数を得るという手法である．

双一次変換では，原理的に変換に誤差が含まれることがわかっており，アナログの周波数領域とディジタルの周波数領域に後述の式(3)のような関係がある．

ここで，式(2)のsはアナログ領域のパラメータであり，$j\omega_a$で表す．z^{-1}はディジタル領域のパラメータであり，$e^{-j\omega_d T}$で表すと，

$$j\omega_a=\frac{2}{T}\frac{1-e^{-j\omega_d T}}{1+e^{-j\omega_d T}}=j\frac{2}{T}\tan\frac{\omega_d T}{2}$$

この両辺をjで割ると，

$$\omega_a=\frac{2}{T}\tan\frac{\omega_d T}{2}\quad\cdots\cdots(3)$$

となる．

式(3)は，ディジタルとアナログの間で，周波数は比例関係ではないことを表している．では，実際にどの程度の誤差があるのかを，それぞれのフィルタで調べてみよう．

● ローパス・フィルタとは

まずは，ローパス・フィルタから見ていくこととする．図3(a)は，アナログのローパス・フィルタである．

この伝達関数は，式(4)で表される．

$$\frac{E_o}{E_i}=\frac{b}{s^2+as+b}$$

$$a=\frac{1}{C_2}\left(\frac{1}{R_1}+\frac{1}{R_2}\right)$$

$$b=\frac{1}{C_1C_2R_1R_2}\quad\cdots\cdots(4)$$

リスト1 Bi-Quadフィルタの設計結果

```
#define A0   0
#define B0   1
#define B2   2
#define A2   3
#define B2   4
#define B3   5
#define C0   1.675603 / 4
#define C1   3.351206 / 4
#define C2   1.675603 / 4
#define D1   0.123324
#define D2   -5.825737 / 8
A_REG = A_REG + B1 * C2;
A_REG = IN_L * 1.0;        ①
A0    = A_REG;             ②
A_REG =         B2 * D1;   ③
MAG(4);                    ④
A_REG = A_REG + A0 * C0;   ⑤
A_REG = A_REG + B0 * C1;
A_REG = A_REG + B1 * C2;
A_REG = A_REG + B2 * D1;
A_REG = A_REG + B3 * D2;   ⑥
A_REG = A_REG + B3 * D2;
A2    = A_REG;             ⑦
OUT_L = A_REG;             ⑧
```

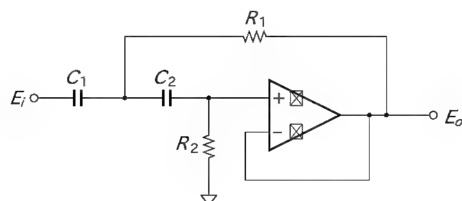
ここで，元となるアナログ・フィルタの設計条件を表1に示す．

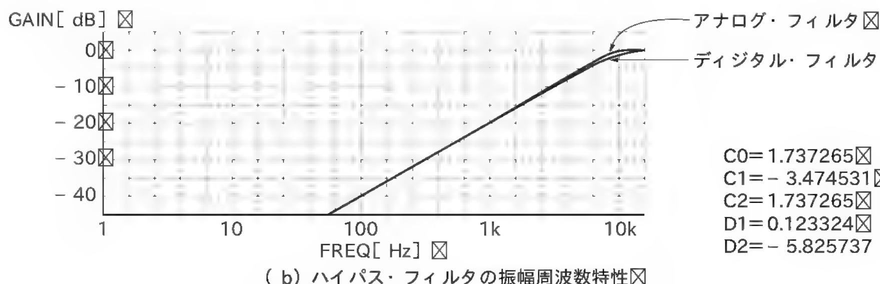
元のアナログ・フィルタと双一次変換後のディジタル・フィルタの振幅周波数特性を図3(b)に示す．図から，周波数が高くなるにつれて，ディジタル・フィルタのほうがより大きく減衰していることがわかる．これは仕様上の不具合といえるが，別な見方をすれば，このような使い方では次数の低いフィルタで，急峻なフィルタを得られたともいえる．一般的には，遮断域の減衰は大きいほうが望ましいので，特に厳密な特性を望む場合以外は，この特性のまま使うことが多い．

ここで設計した結果をリスト1に示す．これは，図2のBi-Quadフィルタで実現したものをxDSP-1のアセンブリ言語で表記したプログラムである．リスト1の概要は，以下のとおりである．

① IN_L(A-D入力の左チャネル)を1.0倍して，A_REGに格納する
② A_REGの内容をA0番地に書き込む
③ B2番地の内容と係数(D1)を乗算し，A_REGに格納する
④ 積和前にあらかじめデータを4倍する．係数が±1.0倍以上になるときに使用する．係数側は4で除算しておく．Mag(4)命令を実行すると，係数を4倍にすることと等価で

New Products ── ACCESS，BREWプラットホーム向け「NetFront for BREW」を開発
(株)アクセスは，情報家電向けWebブラウザNetFrontをQUALCOMM社のBREWプラットホームに対応させた「NetFront for BREW」を開発したことを発表した．

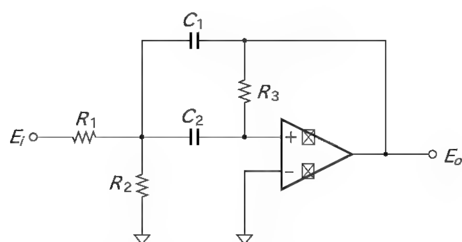


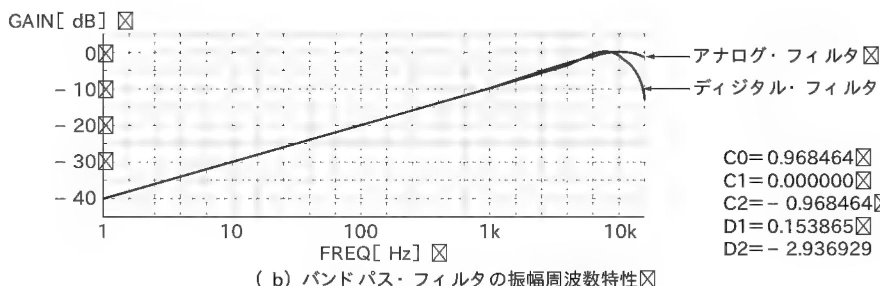


(a) アナログ・ハイパス・フィルタ

(b) ハイパス・フィルタの振幅周波数特性

図4 ハイパス・フィルタ回路





(a) アナログ・バンドパス・フィルタ

(b) バンドパス・フィルタの振幅周波数特性

図5 バンドパス・フィルタ回路

ある．この命令は，これ以降も有効となる

⑤ A0 番地の内容と係数(C0)を乗算し，A_REG を積算する

⑥ 係数(D2)は，±4.0倍を超えているので，同じものを2回積算することで同等の結果を得る．係数側は8で除算しておく

⑦ A_REG の内容を A2 番地に格納する

⑧ OUT_L(D-A 出力の左チャネル)に A_REG の内容を出力する

※ #define 文で，xDSP-1の表現できるより高い係数精度を与えた場合は，7ビットに丸められる．たとえば，1.0という表記であれば，強制的に正の最大値(0.9921875)で置き換えられる

※ IN_L ： A-D の左チャネルを示す予約語
OUT_L： D-A の左チャネルを示す予約語

※ A0： BP の値に A0 を加算したものが実アドレスとなる
B0： BP の値に B0 を加算したものが実アドレスとなる

この例で，A1 番地と A3 番地に書き込む動作が行われていないように見えるが，その理由は後述の「ディレイ・ラインの作成」で詳しく説明する．BP についてもそこで説明する．

● ハイパス・フィルタとは

次に，ハイパス・フィルタについて調べてみる．アナログのハイパス・フィルタを図4(a)に示す．

この伝達関数は，以下の式(5)のように表される．

$$\frac{E_o}{E_i}=\frac{s^2}{s^2+as+b} \quad \cdots\cdots (5)$$

ここで a と b は，

$$a=\frac{1}{R_2}\left(\frac{1}{C_1}+\frac{1}{C_2}\right)$$

$$b=\frac{1}{C_1C_2R_1R_2}$$

である．

元となるアナログ・フィルタの設計条件は，先ほどの表1のとおりであった．

図4(b)にこのハイパス・フィルタの振幅と周波数特性を示す．ローパス・フィルタと同様に，高域で減衰が大きいハイパス・フィルタの場合，その影響が大きくなるので，実用上不具合があることが多い．その場合，プリワーピングという手法であらかじめ補正する．

カットオフ周波数の10kHzを式(2)の ω_d に代入して ω_a を求め，これを元にアナログ・フィルタを設計すればよい．また，ほかのタイプのアナログ・フィルタを使うことで，誤差の影響を減らせる場合もある．

● バンドパス・フィルタとは

図5(a)にアナログのバンドパス・フィルタを示す．この伝達関数は，次の式(6)で表される．

$$\frac{E_o}{E_i}=\frac{as}{s^2+as+b} \quad \cdots\cdots (6)$$

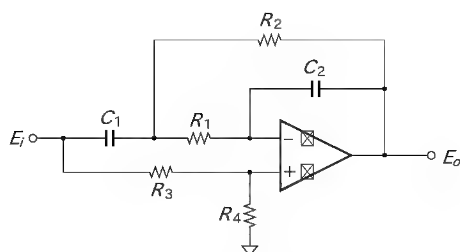
ここで，a と b は以下のとおりである．

$$a=\frac{1}{C_1}\left(\frac{1}{R_1}+\frac{1}{R_2}\right)+\frac{1}{C_2}\left(\frac{1}{R_1}+\frac{1}{R_3}\right)$$

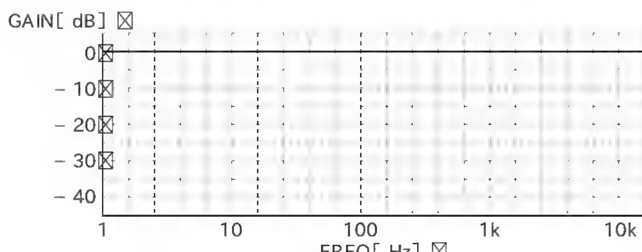
$$b=\frac{1}{C_1C_2R_3}\left(\frac{1}{R_1}+\frac{1}{R_2}\right)$$

元となるアナログ・フィルタの設計条件は，表1のとおりで

Information —— ACCESS，ネットワーク経由で入れ替え可能なユーザ・インターフェース・フレームワークを発表
(株)アクセスは，ネットワーク経由でサービスやプログラムの配信を可能にするユーザ・インターフェース・フレームワーク「NetFront Dynamic Menu」を発表した．NetFrontレンダリング・エンジンをベースに，コンテンツとアプリケーションの統合を可能にする．



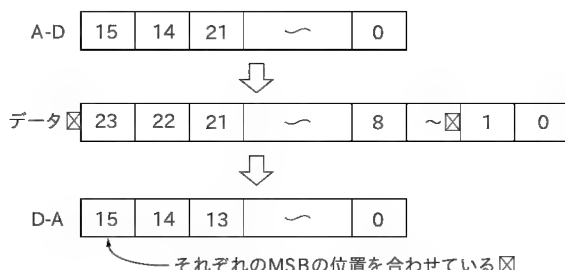
（a）アナログ・オールパス・フィルタ



C0＝1.209958
C1＝1.766526
C2＝1.000000
D1＝−1.766526
D2＝−1.209958

（b）オールパス・フィルタの振幅周波数特性

図6　オールパス・フィルタ回路



（a）A-D，D-Aとデータとの関係

データ　23　22　21　〜　1　0

小数点

2進表記	10進表記
011111111111111111111111	0.99999940..
：	：
000000000000000000000000	0.0
：	：
100000000000000000000000	−1.0

（b）データ語長

図7　データの関係



小数点

2進表記	10進表記
0111111	0.9921875
：	：
0000000	0.0
：	：
1000000	−1.0

図8　係数語長

あった．

このバンドパス・フィルタも，ほかのフィルタと同様に高域で誤差が生じる．

バンドパス・フィルタでは，通過域のカットオフ周波数が重要になることが多いので，その場合には双一次変換は不向きである．その場合，ほかの直接設計法などを使うことが望ましい．

● オールパス・フィルタとは

アナログのオールパス・フィルタを図6に示す．この伝達関数は，次の式(7)で表される．

$$\frac{E_o}{E_i}=\frac{s^2-as+b}{s^2+as+b} \quad \cdots\cdots(7)$$

なお，aとbは，以下のとおりである．

$$a=\frac{1}{C_1}\left[\frac{1}{R_1}+\frac{1}{R_2}\right]$$

$$b=\frac{1}{C_1C_2R_1R_2}$$

元となるアナログ・フィルタの設計条件は，表1のとおりである．

オールパス・フィルタは，振幅周波数特性では誤差はないが，移相器として使う場合には注意が必要である．

● データ語長とデータ範囲の決定

A-D，D-A変換器と内部のデータとの関係を図7(a)に示す．

フィルタの振幅周波数特性は，通過域で0dBであることが多い．つまり，積和の出力が入力より小さくなるので，範囲が$-1.0 \leq d < 1.0$であればよいことになる．符号付きデータなので2の補数の形式とする．

データ語長は，外部メモリとの関係から8ビットの倍数が使いやすいので，24ビットとする．これを図7(b)に図示する．小数点はビット23とビット22の間にある．

● 係数語長と係数範囲の決定

係数の範囲は，データと同様に$-1.0 \leq c < 1.0$で，2の補数の形式とする．

係数語長を7ビットとすると，図8のようになる．小数点はビット6とビット5の間にある．

ここでは使用しないが，係数語長が7ビットで不足な場合は，倍精度乗算も可能である．

● シフト加算による積和の方法

xDSP-1は，ハードウェア開発初心者向け教材という意味合いもあるので，極力簡略化したアーキテクチャである．シフト加算は，我々が筆算で計算するのと同じなので，乗算と積算のしくみが理解しやすいというメリットがある．

シフト加算方式での乗算の動作を図9と式(8)に示す．

News Flash――ルネサス，ソフト・エラーを防止した「superSRAM」を開発
(株)ルネサス テクノロジは，SRAMセルとDRAMキャパシタ技術を融合したメモリ・セル「superSRAM」を開発した．メモリに外部からアルファ線や中性子線が入射したときにメモリの情報が失われる「ソフト・エラー」発生数を従来製品と比較して約4桁改善した．

$$D\times C=\left\{-(1\times d23)+\left(\frac{1}{2}\times d22\right)+\left(\frac{1}{4}\times d21\right)+\cdots\cdots+\left(\frac{1}{2^{23}}\times d0\right)\right\}$$

$$\times\left\{-(1\times c6)+\left(\frac{1}{2}\times c5\right)+\left(\frac{1}{4}\times c4\right)+\cdots+\left(\frac{1}{2^{6}}\times c0\right)\right\}$$

$$=-\left\{-(1\times d23)+\left(\frac{1}{2}\times d22\right)+\cdots+\left(\frac{1}{2^{23}}\times d0\right)\right\}\times c6$$

1回目の部分積…①〔

$$+\left\{-\left(\frac{1}{2}\times d23\right)+\left(\frac{1}{4}\times d22\right)+\cdots+\left(\frac{1}{2^{24}}\times d0\right)\right\}\times c5$$

2回目の部分積…②〔

$$+\left\{-\left(\frac{1}{4}\times d23\right)+\left(\frac{1}{8}\times d22\right)+\cdots+\left(\frac{1}{2^{25}}\times d0\right)\right\}\times c4$$

3回目の部分積…③〔

$$+\left\{-\left(\frac{1}{8}\times d23\right)+\left(\frac{1}{16}\times d22\right)+\cdots+\left(\frac{1}{2^{26}}\times d0\right)\right\}\times c3$$

4回目の部分積…④〔

$$+\left\{-\left(\frac{1}{16}\times d23\right)+\left(\frac{1}{32}\times d22\right)+\cdots+\left(\frac{1}{2^{27}}\times d0\right)\right\}\times c2$$

5回目の部分積…⑤〔

$$+\left\{-\left(\frac{1}{32}\times d23\right)+\left(\frac{1}{64}\times d22\right)+\cdots+\left(\frac{1}{2^{28}}\times d0\right)\right\}\times c1$$

6回目の部分積…⑥〔

$$+\left\{-\left(\frac{1}{64}\times d23\right)+\left(\frac{1}{128}\times d22\right)+\cdots+\left(\frac{1}{2^{29}}\times d0\right)\right\}\times c0$$

7回目の部分積…⑦〔

…………………………(8)

データを D(23： 0)，係数を C(6： 0)とする．D_REGは，クロックごとに右へ1ビットほどシフトするので，シフトを行うごとにその値は半分になる．

図9のように，1クロックあたり，一つの部分積を生成する．式(8)のもつ意味は，

① $c6$が‘1’のときは，$-D$をA_REGに積算する
　‘0’のときは何も積算しない
② $c5$が‘1’のときは，$D/2$をA_REGに積算する
　‘0’のときは何も積算しない
③ $c4$が‘1’のときは，$D/4$をA_REGに積算する
　‘0’のときは何も積算しない
④ $c3$が‘1’のときは，$D/8$をA_REGに積算する
　‘0’のときは何も積算しない
⑤ $c2$が‘1’のときは，$D/16$をA_REGに積算する
　‘0’のときは何も積算しない
⑥ $c1$が‘1’のときは，$D/32$をA_REGに積算する
　‘0’のときは何も積算しない
⑦ $c0$が‘1’のときは，$D/64$をA_REGに積算する
　‘0’のときは何も積算しない

ということであり，結局 C(6 : 0)の値により，部分積をA_REGに「足す」か「足さないか」を選択することになる．

①～⑦の演算を7回に分けて行うと，求める乗算結果が生成される．

言い換えると，7クロックで1回の乗算が完了する．実際には1クロックの休止を含むので，8クロックを要する．

xDSP-1のシフト加算のVHDL表記をリスト2に示す．

● オーバフロー対策

いくつかのデータを積算すると，その表現できる最大値を超えてしまうことがある．これをオーバフローと呼んでいる．

xDSP-1の場合は，$-1.0 \leq d < 1.0$の範囲に収まらない場合をオーバフローとしている．

この対策として，積和の直前にあらかじめ定数で除算（プリ・スケーリング）し，積和の直後に同じ定数を乗算（ポスト・スケーリング）する手法をとる．具体的には，その定数を16とする．その理由は，4ビット左（右）シフト演算で実現できることと，実際のアプリケーションにおいて積和数は10以下であることが多いからである．

一連の積和の期間にはオーバフローを防ぐことができても，ポスト・スケーリング後はやはりオーバフローを起こす可能性がある．その場合は，クリッピング処理が必要になる．オーバフローが生じた場合，正負それぞれの最大値に固定する．

xDSP-1のクリッパのVHDL表記をリスト3に示す．

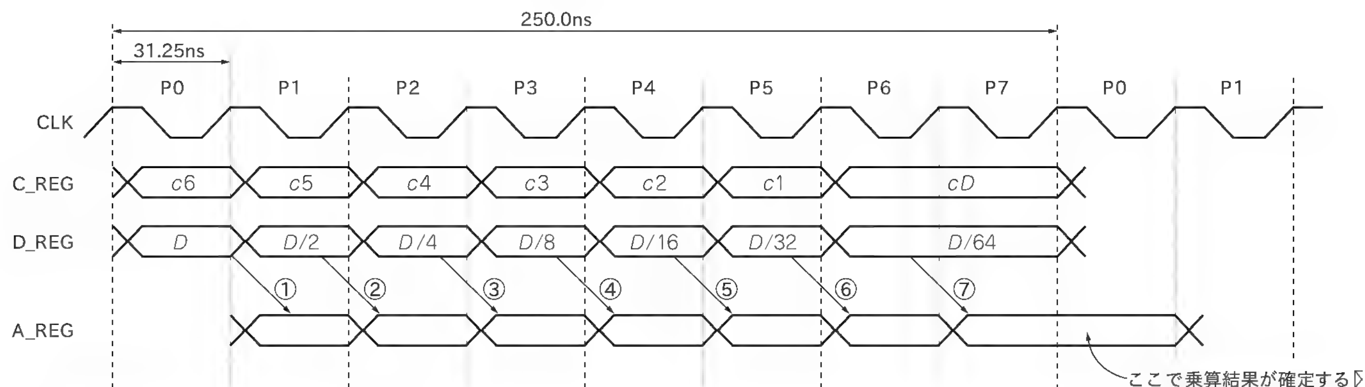


図9　シフト加算のタイミング・チャート

New Products――無線LAN対応組み込みデバイス・サーバ「WiPort」
（株）日新システムズは，無線LANに対応した組み込みデバイス・サーバ「WiPort」を発売した．同社の「XPort」の無線対応版という位置付けの製品で，シリアル・ポート×2チャネルをもち，TCP/IPプロトコル・スタックによりWebブラウザでリモート管理が行える．

リスト 2　xDSP-1のシフト加算（VHDL表記）

```
process(CLK)
variable    d_reg28        : std_logic_vector(27 downto 0);
variable    minus_d_reg28  : std_logic_vector(27 downto 0);
begin
    if (CLK'event and CLK='1') then
        if (PHASE = P7) then
            if (DBL = '1') then
                D_REG(23 downto 0) <= D_REG(23 downto 0);          -- 倍精度演算のときは，D_REGを更新しない
            else
                D_REG(23 downto 0) <= DATA_RAM;                    -- DATA_RAMの内容をD_REGにロードする
            end if;
        else
            D_REG(23 downto 0) <= D_REG(23) & D_REG(23 downto 1);  -- 算術右シフト
        end if;

        d_reg28 := D_REG(23) & D_REG(23) & D_REG(23) & D_REG(23) & D_REG;  -- プリ・スケール（1/4にする）

        case PHASE is
        when P0 =>                                                 -- P0フェイズ
            minus_d_reg28 := not d_reg28 + '1';                    -- minus_d_reg28 = d_reg28 * -1.0
            if (CLR = '1') then                                    -- 積和の開始時は，まずA_REGをクリアする
                if (C_REG(6) = '1') then                           -- 係数が負のときは，minus_d_reg28を積算する
                    A_REG <= minus_d_reg28;
                end if;
            else
                if (DBL = '1') then
                                  -- 倍精度演算のときは，
                    A_REG <= d_reg28;
                                  -- 係数が負であっても-1.0倍しない
                else
                    if (C_REG(6) = '1') then
                        A_REG <= A_REG + minus_d_reg28;
                    end if;
                end if;
            end if;
            PHASE <= P1;
        when P1 =>                             -- P1フェイズ
            if (C_REG(5) = '1') then
                A_REG <= A_REG + d_reg28;
            end if;
            PHASE <= P2;
        when P2 =>                             -- P2フェイズ
            if (C_REG(4) = '1') then
                A_REG <= A_REG + d_reg28;
            end if;
            PHASE <= P3;
        when P3 =>                             -- P3フェイズ
            if (C_REG(3) = '1') then
                A_REG <= A_REG + d_reg28;
            end if;
            PHASE <= P4;
        when P4 =>                             -- P4フェイズ
            if (C_REG(2) = '1') then
                A_REG <= A_REG + d_reg28;
            end if;
            PHASE <= P5;
        when P5 =>                                             -- P5フェイズ
            if (C_REG(1) = '1') then
                A_REG <= A_REG + d_reg28;
            end if;
            PHASE <= P6;
        when P6 =>                                             -- P6フェイズ
            if (C_REG(0) = '1') then
                A_REG <= A_REG + d_reg28;
            end if;
            PHASE <= P7;
        when P7 =>                                             -- P7フェイズ
            C_REG <= INST_RAM(9 downto 3);
                        -- 次回の乗算用の係数をINST_RAMからロードする．
                        -- A_REGの内容は更新しない．
            PHASE <= P0;
        when others =>
            PHASE <= P0;
        end case;
    end if;
end process;
```

リスト 3　xDSP-1のクリッパ（VHDL表記）

```
process(A_REG, NOV)
begin
    if (A_REG(27 downto 23) = "00000") then            -- A_REGの内容が正の最大値より小さいとき
        CLIP_OUT <= A_REG(23 downto 0);                -- ポスト・スケール（4倍にする）
    elsif (A_REG(27 downto 23) = "11111") then         -- A_REGの内容が負の最大値より大きいとき
        CLIP_OUT <= A_REG(23 downto 0);                -- ポスト・スケール（4倍にする）
    else
        if (NOV = '1') then                            -- クリップ処理を禁止しているときは，そのまま出力
            CLIP_OUT <= A_REG(23 downto 0);            -- ポスト・スケール（4倍にする）
        else
            if (A_REG(27) = '1') then
                CLIP_OUT <= "100000000000000000000000"; -- 負にオーバフローしたら，負の最大値に張り付ける
            else
                CLIP_OUT <= "011111111111111111111111"; -- 正にオーバフローしたら，正の最大値に張り付ける
            end if;
        end if;
    end if;
end process;
```


New Products――タンベルグデータ，第2世代小型テープ・ストレージ・ソリューション「Tandberg Storage Solution LTO2」を発売
タンベルグデータ（株）は，圧縮時記憶容量400Gバイト（非圧縮200Gバイト），圧縮時記録速度70Mバイト/s（非圧縮35Mバイト/s）の小型テープ・ストレージ・ソリューション「Storage Solution LTO2」を発売した．http://www.tandberg.co.jp/

演算結果をクリップ処理した場合，フィードバック系が不安定になるおそれがあるので，極力オーバフローしないようなアルゴリズムの作成を心掛ける必要がある．

● ディレイ・ラインの作成

前項のフィルタの例では，1サンプルのディレイ・ラインしか使用していない．しかし，オーディオ・エフェクタでは数サンプルから数千サンプルのディレイ・ラインが必要な場合が多いので，効率の良いメモリ・アクセスを心掛けなければならない．具体的には，データ・メモリをリング・バッファとみなし，ディレイ・ラインを作る方法をとる（図10）．図10のBPとは，1サンプルあたり，1ずつデクリメントするカウンタである．BPの値とアドレスを加算したものが，実アドレスとなる．

ここで，A，B，C…の信号列に対し，2サンプル分の遅延を持たせる場合を考える．具体的には，あるサンプルにメモリに書き込んだデータを2サンプル後に読み出すことで実現する．

（a）BPの値は3であるので，書き込みポインタは5番地，読み出しポインタは7番地となる
（b）BPの値は2であるので，書き込みポインタは4番地，読み出しポインタは6番地となる
（c）BPの値は1であるので，書き込みポインタは3番地，読み出しポインタは5番地となる

ここで初めて，2サンプル前のデータが読み出せたことになる．

3 エフェクタを構成するためのアルゴリズム

ここでは，代表的なエフェクタの構成とアセンブリ言語について解説していく．

● アッテネータ

図11にアッテネータのフローを，リスト4にアセンブラ・ソースを示す．

図11からわかるように，入力データに定数（EL）を掛けて出力するだけのものであるが，これでも立派なエフェクタである．アナログのボリュームに相当する．

● ショート・ディレイ

図12（a）にショート・ディレイのフローを示す．

これは，比較的単純な構成で，入力データとその遅延データを加算して出力するものである．周波数特性が図12（b）のような形状になるので，Comb（くし形）フィルタともいう．遅

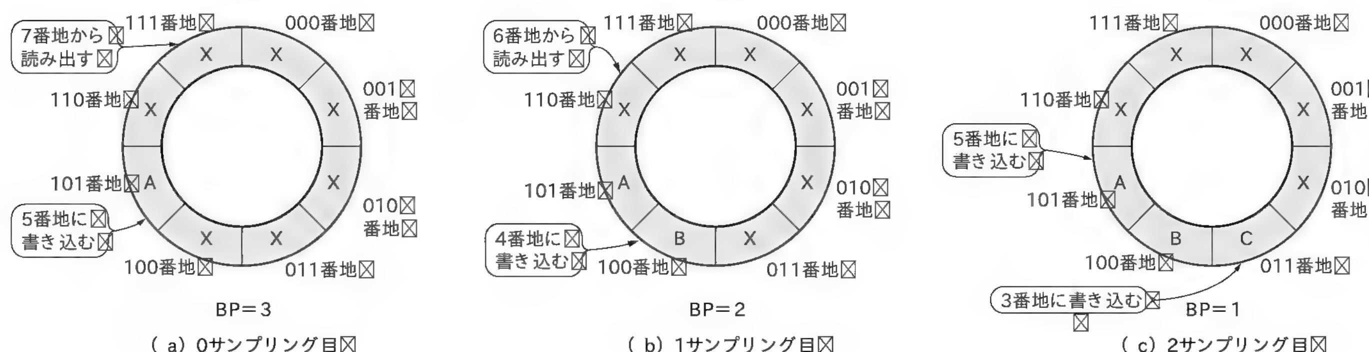


図10　リング・バッファの構成



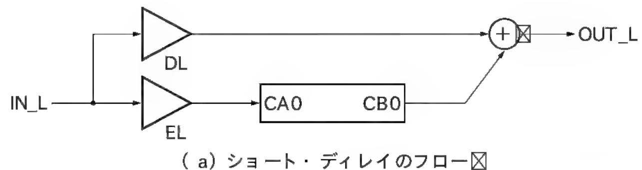
図11
アッテネータのフロー

リスト4　アッテネータ

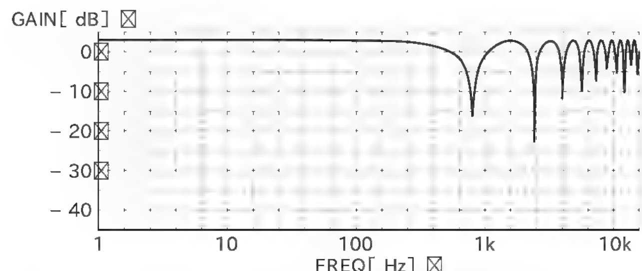
```
A_REG = IN_L * EL;
OUT_L = A_REG;
```

リスト5　ショート・ディレイ

```
A_REG = IN_L * 1.0;
A     = A_REG;
A_REG = IN_L * C0;
A_REG = A_REG + B * C1;
OUT_L = A_REG;
```



（a）ショート・ディレイのフロー



（b）ショート・ディレイの振幅周波数特性

図12　ショート・ディレイ

Information ――エニアとYDCが提携し，OSEがARM920T搭載評価ボードA-platをサポート
エニア・エンベデッド・テクノロジー（株）と横河ディジタルコンピュータ（株）（以下YDC）は，YDCのA-plat（Freescale社製i.MX1 ARM920T搭載標準評価ボード）をエニアのリアルタイムOS「OSE」がサポートし，A-plat＋OSEのソリューションを提供するために提携することを発表した．

延量は数10ms程度である．ボーカルやギター・ソロによく使われる．

これのアセンブラ・ソースをリスト5に示す．

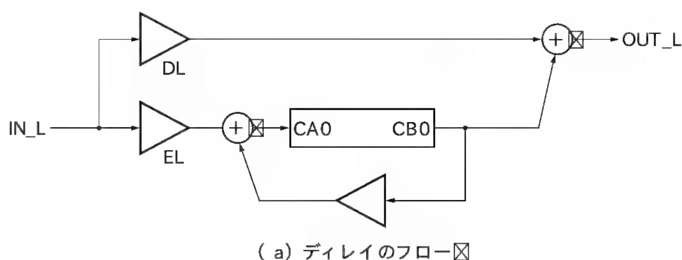
● ディレイ（エコー）

ディレイとは，カラオケなどでおなじみのエコーである．図13（b）のようなインパルス・レスポンスをもつ．図13（a）のCB0－CA0にサンプリング周期をかけた時間が，ディレイ・タイムに相当する．通常，数百msのディレイ・タイムで使用される．

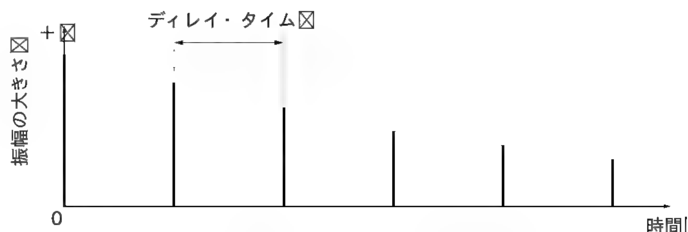
アセンブラ・ソースをリスト6に示す．

リスト6　ディレイ

```
A_REG = IN_L * EL;
A_REG = A_REG + CB0 * FB0;
CA0   = A_REG;
A_REG = IN_L * DL;
A_REG = A_REG + CB0 * 1.0;
OUT_L = A_REG;
```



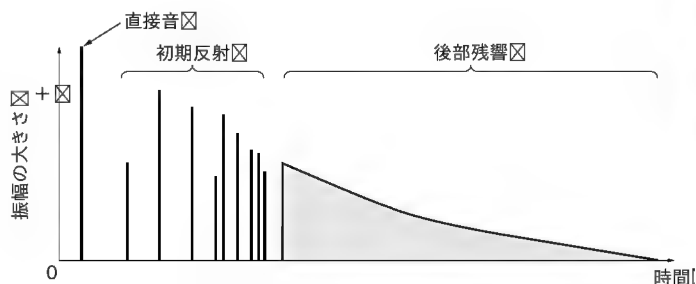
（a）ディレイのフロー



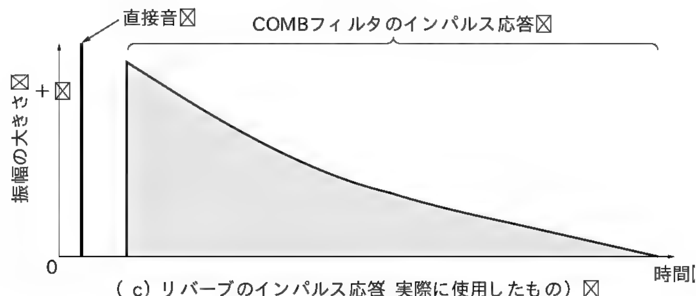
（b）ディレイのインパルス応答

図13　ディレイ

（a）リバーブのフロー図



（b）リバーブのインパルス応答（実際のホール残響）



（c）リバーブのインパルス応答（実際に使用したもの）

図14　リバーブの構成

New Products――エクセルソフト，Jungo社製のLinuxデバイス・ドライバ開発ツール「WinDriver Linux PowerPC」を販売開始
エクセルソフト（株）は，Jungo社が開発したPowerPC対応のLinuxデバイス・ドライバ開発ツール「WinDriver Linux」の販売を開始した．PCIバス版とUSB版の二つを用意している．本ツールでは，OS内部の知識やカーネル・レベルのプログラミングが不要．

COLUMN

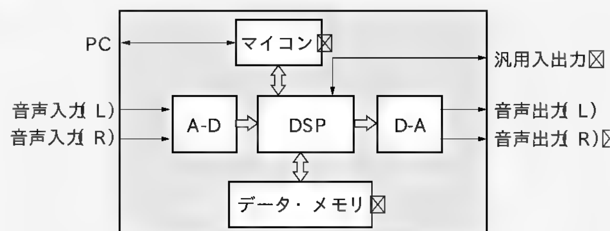
xDSP-1の概要

ここで，第3章で使用したxDSP-1について，簡単に説明する．
<特徴>
● 内部アーキテクチャを公開
DSPが汎用FPGAに実装されている．VHDLソースが公開されているので，ユーザによるカスタマイズが可能．
● サンプル・プログラムを公開
エフェクタのサンプル・プログラムを収録している．ユーザが独自のエフェクタを作ることもできる．
● 添付ソフトウェア
- xDSP-1アセンブラ
- プログラム・ローダ
- FPGAコンフィグレーション・ツール

● 音声信号処理に最適なアーキテクチャ
- ハーバード・アーキテクチャを採用しており，効率よく音声エフェクタを実現できる．5～6台のマルチエフェクトが実装可能

● PCからプログラミング可能
信号処理を中断することなく，DSPプログラムの書き換えが可能．
● 2チャネルのA-D，D-A変換器を実装
左右のチャネルが同位相になるように補正している．
● 汎用入出力用コネクタを実装
40ビットをユーザ定義のI/Oとして使用できる．
<本章で紹介した以外の演算機能>
間接アドレシング，条件分岐，倍精度乗算など
●実現可能なエフェクタ
コーラス，フランジャ，フェイザ，タッチ・ワウ，ピッチ・シフタ，ディストーション，コンプレッサ，サウンド・ジェネレーションなど
<発売元>
（株）ゼクー
http://www.xecoo.co.jp/
<発売予定>
2005年2月
<予定価格>
19,800円（税込）

<仕様>
演算数/sample：125回
1演算時間：250ns
サンプリング周波数：32kHz
入力（A-D）：2チャネル（16ビット）
入力レベル：2Vp-p
S/N（A-D）：89dB
出力（D-A）：2チャネル（16ビット）
出力レベル：2Vp-p
S/N（D-A）：92dB
RS-232-C：1チャネル
汎用入出力：40ビット
電源：DC 6V
基板サイズ：
100mm×125mm



図A　xDSP-1の構成

リスト7　リバーブ

```
A_REG = IN_L * EL;
A_REG = A_REG + CB0 * FB0;
CA0   = A_REG;

A_REG = IN_L * EL;
A_REG = A_REG + CB1 * FB1;
CA1   = A_REG;

A_REG = IN_L * EL;
A_REG = A_REG + CB2 * FB2;
CA2   = A_REG;

A_REG = IN_L * EL;
A_REG = A_REG + CB3 * FB3;
CA3   = A_REG;

A_REG = CT0  * 1.0;
A_REG = A_REG + CT1 * 1.0;
A_REG = A_REG + CT2 * 1.0;
A_REG = A_REG + CT3 * 1.0;
A_REG = A_REG + AB0 * AFB0;
AA0   = A_REG;

A_REG = AA0 * AFF0;
A_REG = A_REG + AB0 * 1.0;
A_REG = A_REG + AB1 * AFB1;
AA1   = A_REG;

A_REG = AA1 * AFF1;
A_REG = A_REG + AB1 * 1.0;
A_REG = A_REG + IN_L * DL;
OUT_L = A_REG;
```

● リバーブ

リバーブは，部屋やホールの残響を電子的にシミュレーションするものである．実際のホール残響のインパルス応答を簡略化して図14（b）に示す．

ここでは「Schroederの系」と呼ばれるアルゴリズムを紹介する．構成としては，ディレイをいくつも組み合わせたようなものである．

この例では簡略化のため，図14（c）のように初期反射部分は省いている．後段のオールパス・フィルタは残響の密度を上げる目的で使われている．

パラメータの決定については，リバーブ特有の考慮すべき点がある．自然な残響感を実現するためには，各後部残響部の減衰包絡を一致させる必要がある．

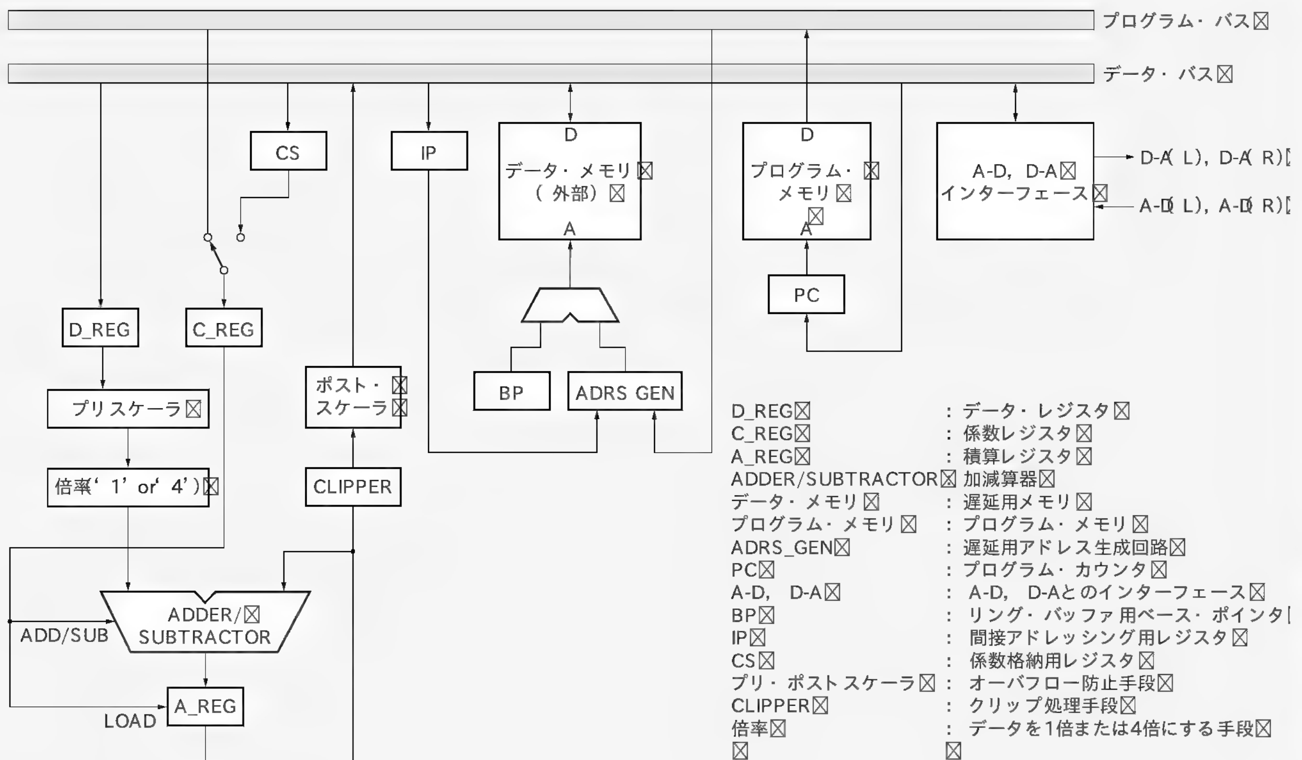
減衰包絡が60dB減衰する時間をリバーブ・タイムという．このリバーブ・タイムは，式（9）から求めることができる．

$$T_{rev} = \frac{60mT}{-20\log_{10} g}$$

g：フィードバック・ゲイン．$FB0$～$FB3$がこれにあたる．
mT：遅延時間（s）
T_{rev}：リバーブ・タイム（s）

図14（a）の例では，次式が成り立てばよいことになる．

Information ── Novell，オープン・ソースの開発プラットホーム「Mono 1.0」の提供を開始
米Novell社は，.NETフレームワークに基づくオープン・ソースの開発プラットホーム「Mono 1.0」の提供を開始した．Mono 1.0は，C#コンパイラと.NET互換のランタイム，二つのAPIを備えている．Linuxやクロス・プラットホームの効率的な実現を図れる．



図B　xDSP-1のアーキテクチャ

$$\frac{60(CB0-CA0)T}{-20\log_{10}FB0}=\frac{60(CB1-CA1)T}{-20\log_{10}FB1}=\frac{60(CB2-CA2)T}{-20\log_{10}FB2}=\frac{60(CB3-CA3)T}{-20\log_{10}FB3} \quad \cdots\cdots\cdots\cdots\cdots(9)$$

リバーブのアセンブラ・ソースをリスト7に示す．

ここに紹介したものはもっとも原始的なもので，今ではいろいろなアルゴリズムが考案されている．各メーカとも，空間的な広がりを豊かに表現するようにくふうしているようである．

● トレモロ

トレモロは，音声の振幅を周期的に変化させるもので，もっとも古典的なエフェクタである．図15(a)にフローを示す．

この例では，図15(b)のような三角波で振幅を変調しているが，正弦波で変調したほうがなめらかな効果があるかもしれない．出力信号は図15(c)のようになる．

このアセンブル・ソースをリスト8に示す．

以下は，リスト8の大まかな説明である．

① クリップ処理禁止

リスト8　トレモロ

```
D_CLIP;                    ①
A_REG = B * 1.0;
A_REG = A_REG + CONST0;
A     = A_REG;             ②
E_CLIP;                    ③

A_REG = ABS(A) * AMP;      ④
A_REG = A_REG + CONST1;    ⑤
CS0   = A_REG;             ⑥

A_REG = IN_L * CS0;
OUT_L = A_REG;
```

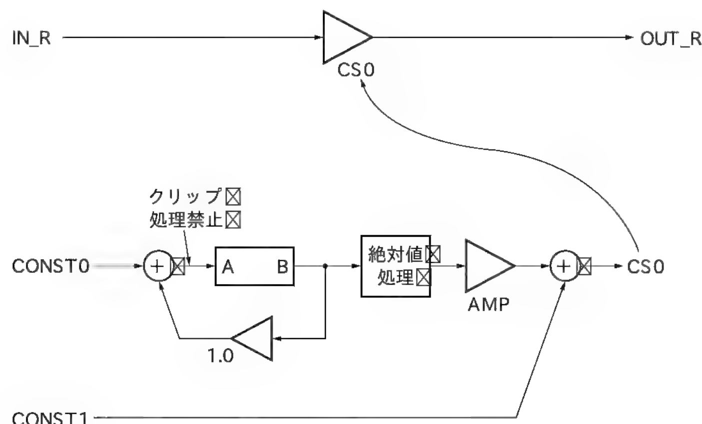
② オーバフローを許しているので，CONST0を1サンプルあたりの増分とするのこぎり波が生成される．負の最大値から正の最大値までを振幅する

③ クリップ処理許可

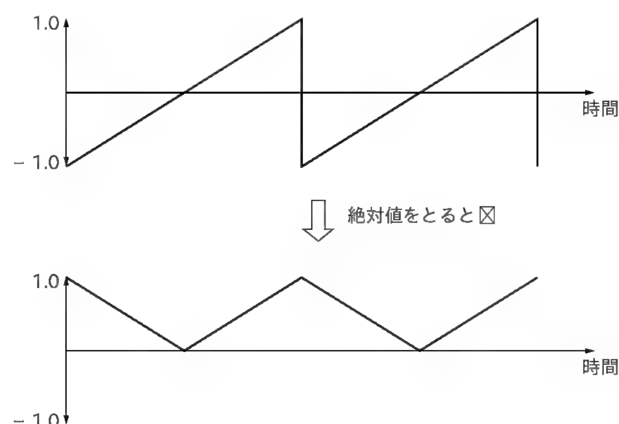
④ ABS(A)は，Aの内容の絶対値を得る． 図15(b)のように，三角波が生成される

⑥ CS0は係数を格納するレジスタである．xDSP-1はCS0の

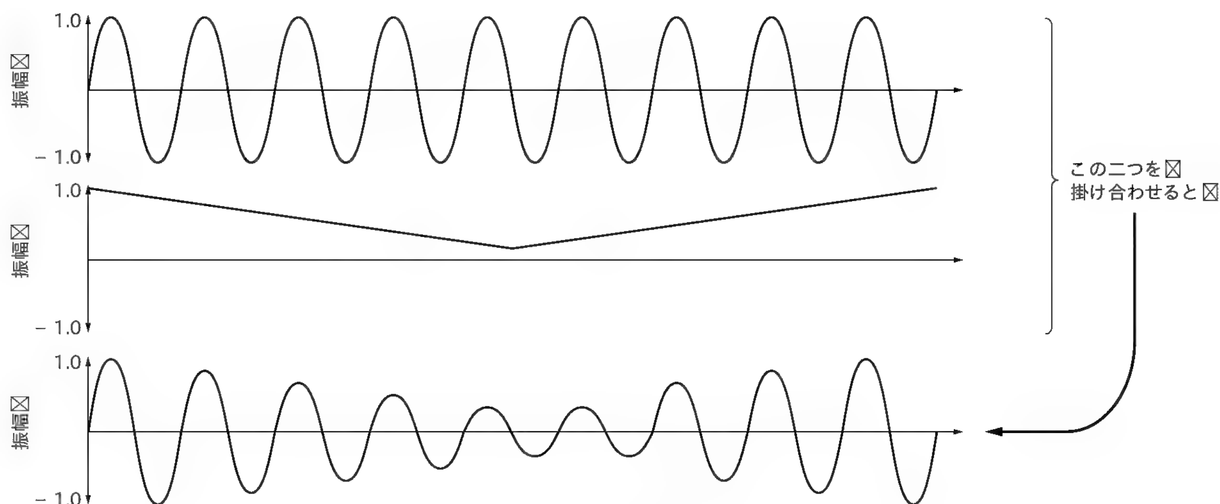
Information ── Microsoft，Windows CE5.0 英語版の製造を開始
米Microsoft社は，Windows CE5.0(開発コード：Macallan)の英語版の製造を7月9日から開始した．60個以上のデバイス・ドライバ・サンプルやリモートからのデバイス・パフォーマンス・モニタリング，Direct3Dモバイルなどの機能が追加されている．



(a) トレモロのフロー



(b) 三角波の生成



(c) トレモロの振幅波形

図15 トレモロの構成

内容を係数として使用することができる

おわりに

汎用DSP開発には検討すべき項目が山積しており，本章で紹介できたのは，その一部にすぎない．しかし，開発の進め方はわかっていただけたと思う．

ここで紹介したエフェクタをシミュレートするアプリケーションを，本誌のWebサイトからダウンロードできるので，これを用いて“感じ”をつかんでいただきたい．

参考文献
(1) 畔津明仁；はじめての数値演算回路設計，インターフェース，1990年12月号，CQ出版．
(2) 谷本茂；オペアンプ実戦技術，pp.164-170，誠文堂新光社，1979．
(3) 越山常治・山崎芳男；PCM/ディジタル・オーディオのすべて，pp119，誠文堂新光社，1980．

のざわ・なおや (株)ゼクー

I/F ESSENCE 好評発売中
よくわかるディジタル画像処理
フィルタ処理からDCT&ウェーブレットまで
貴家 仁志 著
A5判 232ページ 3.5"2DD FD付き
定価2,854円(税込)
ISBN4-7898-3677-0
CQ出版社 〒170-8461 東京都豊島区巣鴨1-14-2 販売部 TEL.03-5395-2141 振替 00100-7-10665

Information ――イーソルのμITRON4.0準拠のRTOSとFATファイル・システムが指紋認証組み込み機器のソフトウェア部に採用
イーソル(株)のμITRON4.0準拠のRTOS「PrKERNELv4」とFATファイル・システム「PrFILE」が，富士通コンピュータテクノロジーズの「指紋認証組込みソリューションズ」のソフトウェア部に採用された．

4 2次元FFTを使ったディジタル・フィルタ
画像処理アプリケーションの作成

本章では，画像処理アプリケーションの作成を通して，そのアルゴリズムの元となる1次元のフーリエ変換から離散フーリエ変換，そして2次元のフーリエ変換までを解説する．また，作成したアプリケーションは，本誌のWebサイトからダウンロードできる．
（編集部）

門屋 純一

はじめに

昨今では，カメラ付き携帯電話や安価で高解像度のディジタル・カメラの普及にともない，ディジタル画像は一般にもなじみ深いものとなった．さらに，コンピュータの高速化，記憶装置の大容量化などと相まってディジタル画像処理は娯楽から産業分野まで，ますます身近になりつつある．

本章ではおもにPCを使ったディジタル画像処理について解説する．画像処理の基礎としてFA分野でもよく使われる画像フィルタを解説し，PCで作成したプログラムでその効果を確認する．

次に直交変換の一例としてフーリエ変換を取り上げる．画像処理への応用としてのプログラムを実際にC言語で作りながら2次元の高速フーリエ変換の性質を調べ，さらに簡単な図形認識をPCで試みる．本章で作成したプログラムと全ソース・リストは，章末に掲載したURLからダウンロードできるので，ぜひ実際のプログラムに触れ，試していただきたい．

また，ディジタル画像処理でよく使う用語と基本的な概念について，章末で解説しているので参照されたい．

1 ディジタル画像処理の手段とPC

画像処理はカメラやビデオ・カメラ，スキャナなどの入力装置から取り込んだ画像を目的に応じて変換する作業であり，その手段としては，

1) 専用ハードウェアによる処理

2) 汎用CPUによるソフトウェア演算

などが考えられる．それぞれのメリットとデメリットを考えてみると，まず，専用ハードウェアを使うメリットはその高速性である．PCなどに使われている多くのCPUでは，命令の取り込みとデータの読み書きを同じバスを使って行うため，実行に時間がかかる．DSPや専用LSIの多くでは，命令とデータを同時にアクセスできるハーバード・アーキテクチャが採用されており，命令フェッチ・サイクルが省略されて高速処理が可能になっている．製品によってはマルチポートRAMを使って，一つのメイン・メモリでハーバード・アーキテクチャと同じ効果をあげるものも存在する．また，DSPでは積和演算（かけ算した結果を加算するという演算）が多いので，それを1クロックで行える命令などを備えている．

PCのソフトウェアによる画像処理では，ハードディスクなど2次記憶メディアからファイルを入力する場合など，専用ハードウェアとは違うケースで使われることが多く見られるが，専用ハードウェアを設計する前段階でハードウェアのシミュレーションを行ってアルゴリズムを検証するような際にも有効である．

ネットワーク通信環境や豊富な記憶メディア，出力装置に簡単にアクセスできるというメリットも大きい．また，ソフトウェアの場合，プログラムの任意の位置でブレークをかけたり，デバッグ情報をダンプすることが容易なので，専用のハードウェアと比べてデバッグ効率に優れるケースが多いと考えられる．

専用ハードウェアに比べて低速とはいっても，CPUが大容量のキャッシュを備えている場合などは，キャッシュにヒットし続ければ高速化が望めるし，Intel社のPentiumプロセッサのMMX，SSE命令のように複数の積和演算を同時に行うことで，DSPに迫る高速演算も場合によっては可能である．

2 画像フィルタの構成法

画像フィルタは，娯楽や出版分野では一定の視覚効果を与えるために，産業分野ではノイズの除去やパターン認識のための特徴抽出などのために頻繁に使われている．

簡単な画像フィルタは，処理するピクセルの近傍のピクセルに対して重み係数を付けた加重平均をとるだけで実現できる．

M_{11}	M_{12}	M_{13}
M_{21}	M_{22}	M_{23}
M_{31}	M_{32}	M_{33}

図1　フィルタ・マスク

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

図2　平滑フィルタ

−1	0	1
−1	0	1
−1	0	1

図3　水平方向1次微分

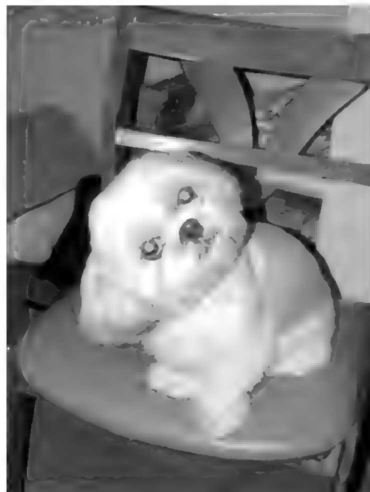


写真1　元の画像

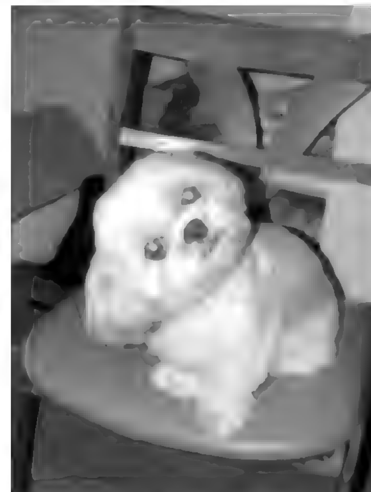


写真2　1次平滑

この重み係数のマトリクスのことを，フィルタ・マスクなどと呼んでいる．

画像処理においては，3行×3列のマスクが多用される（図1）．3×3の魔方陣の中心が処理するピクセルの位置になり，その周辺のピクセルの値に一定の係数を付けて加算したもので，当該ピクセルを置き換えることでフィルタが実現できる．

座標（i, j）のピクセル値を$A(i, j)$，フィルタ演算出力を$B(i, j)$で表すと，一般式は次のようになる．

$$\begin{aligned}B(i, j) = &M11 \times A(i-1, j-1) + M12 \times A(i, j-1)\\ &+ M13 \times A(i+1, j-1) + M21 \times A(i-1, j)\\ &+ M22 \times A(i, j) \quad + M23 \times A(i+1, j)\\ &+ M31 \times A(i-1, j+1) + M32 \times A(i, j+1)\\ &+ M33 \times A(i+1, j+1)\end{aligned}$$

iは水平方向，jは垂直方向の座標で，左上端を原点とすれば，$A(i-1, j-1)$は左上のピクセルを，$A(i+1, j+1)$は右下のピクセルを表している．

この魔法陣の値（係数$M11$～$M33$）を変えることで，いろいろな性質をフィルタに持たせることができる，まさに魔方陣なのである．

前章で解説があった音声のFIR型ディジタル・フィルタは1次元時間軸での畳み込み演算であったが，それを2次元に拡張して畳み込み係数を極限まで短くしたものと考えることができる．

もちろん，係数の行列をもっと長くすればフィルタの自由度は広がるが，ほぼ長さの2乗に比例して演算時間が増大してしまう．画像も一方向にその輝度の増減をたどっていくと，傾斜のきつい部分や，グラデーションのように傾斜のなだらかな部分があり，1次元の音声波形などと同様に扱うことができる．急激に値の変化する点は周波数が高いと考えられ，なだらかなうねりは周波数が低いと表現することができる．

● 1次平滑フィルタ

では，はじめに図2のようなマトリクスを考えてみよう．これは，中心となるピクセル値と周囲のピクセル値を足し合わせて平均したものに置き換える，平滑フィルタである．

$$\begin{aligned}B(i, j) = &0.1 \times A(i-1, j-1) + 0.1 \times A(i, j-1)\\ &+ 0.1 \times A(i+1, j-1) + 0.1 \times A(i-1, j)\\ &+ 0.2 \times A(i, j) \quad + 0.1 \times A(i+1, j)\\ &+ 0.1 \times A(i-1, j+1) + 0.1 \times A(i, j+1)\\ &+ 0.1 \times A(i+1, j+1)\end{aligned}$$

これを写真1の画像のすべてのピクセルに対して演算した結果は，写真2のように全体がソフトなトーンとなる．

● 1次空間微分フィルタ

それでは，図3のマトリクスを摘要すれば，どのような効果が得られるであろうか．

これは，水平方向の1次空間微分フィルタ（Prewittのオペレータ）と呼ばれるもので，魔法陣の真ん中のカラムがすべてゼロである．仰々しい感じの名前ではあるが，ディジタル信号処理では微分は差分に置き換えられるので，すべて積和演算（乗算と加算）で賄うことができる．

$$\begin{aligned}B(i, j) = &-1 \times A(i-1, j-1) + 0 \times A(i, j-1)\\ &+ 1 \times A(i+1, j-1) + (-1) \times A(i-1, j)\\ &+ 0 \times A(i, j) + 1 \times A(i+1, j)\\ &+ (-1) \times A(i-1, j+1) + 0 \times A(i, j+1)\\ &+ 1 \times A(i+1, j+1)\end{aligned}$$

このフィルタでは，横方向の変化の度合いが大きいところが強調されるので，縦線のエッジ抽出に利用することが可能である．結果は写真3を参照されたい．

理解を深めるために，具体的な値を使って1次元でこのフィルタを考えてみよう．横方向のピクセル数が10あって，その輝度が図4のAのように変化していたとする．かりに0が黒レベ

New Products――コンテック，ファンレス組み込み向けパネル／ボックス・コンピュータ「IPC 620シリーズ」を発売
（株）コンテックは，ファンレス稼動の組み込み向けコンピュータ「パネルコンピュータ」と「ボックス・コンピュータ」の620シリーズを発売した．CPUとしてUltra Low Power Celeron 400MHzを備えている．価格は￥220,500から．


写真3　水平方向1次微分

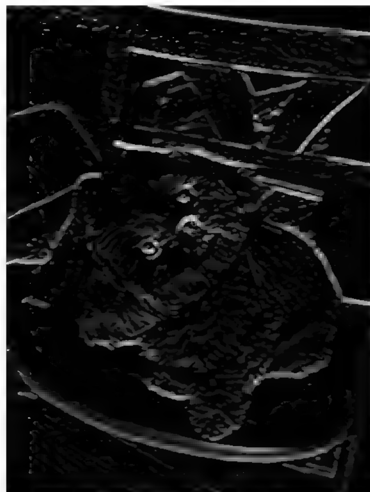

写真4　垂直方向1次微分


写真5　水平方向Sobel

ルで，10が白レベルであったとする．1次元なので，これは(−1, 0, 1)の係数を各スロットごとに積和して値を計算してみよう．

$$B(i) = -1 \times A(i-1) + 0 \times A(i) + 1 \times A(i+1)$$
$$= -A(i-1) + A(i+1)$$

$i=1$の値は$-0+0=0$
$i=2$の値は$-0+10=10$
$i=3$の値は$-0+10=10$
$i=4$の値は$-10+10=0$
$i=5$の値は$-10+10=0$
$i=6$の値は$-10+10=0$
$i=7$の値は$-10+0=-10$
$i=8$の値は$-10+0=-10$

というぐあいに図4のBが計算できる．ちょうどAの立ち上がりと立ち下がりの前後でBの値が変化しているのがわかるだろう．変化のある部分でのみ値をもつのが微分フィルタと呼ばれるゆえんである．

図3のマスクを縦横入れ替えたものが図5の垂直方向1次微分フィルタである．

水平1次微分フィルタから容易に想像できるように，これは縦方向の変化点を強調，つまり横線の抽出に有効である．写真1の原画にこのフィルタを適用した結果を写真4に示す．

この水平方向，垂直方向のフィルタを組み合わせてエッジ抽出を行うには，各ピクセルの$\sqrt{Bh^2+Bv^2}$をとることによって1次空間微分の大きさを得る．

ここで，Bhは水平方向1次微分フィルタの実行結果，Bvは垂直方向1次微分フィルタの実行結果とする．

● Sobelフィルタ

1次微分フィルタで中心点に近いところの係数の重みを増したものをSobelのオペレータと呼ぶ．重みが2倍になっている分だけ微分値も2倍になるので，エッジがより強調される．

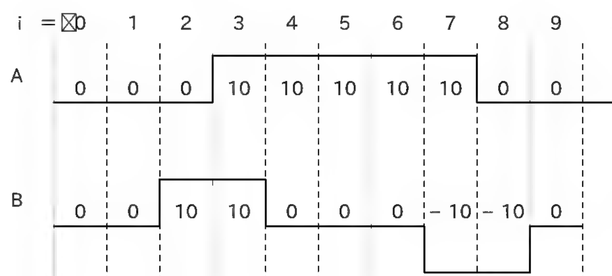


図4　輝度の変化の例

−1	−1	−1
0	0	0
1	1	1

図5　垂直方向1次微分

−1	0	1
−2	0	2
−1	0	1

図6　水平方向Sobel

−1	−2	−1
0	0	0
−1	−2	−1

図7　垂直方向Sobel

Prewittのオペレータと同様に，水平方向(図6)，垂直方向(図7)の演算結果を組み合わせて絶対値をとることでエッジ抽出を行うことができる．原画(写真1)の水平，垂直Sobelフィルタによる演算結果は写真5のようになる．

● 2次空間微分フィルタ

1次空間微分フィルタは，隣り合った画素の大きな輝度変化のある部分を検出することで，輪郭を抽出しようとしたフィルタであった．しかし，通常の画像における輪郭は劇的な輝度変化がある部分とは限らず，そういう画像では1次空間微分フィルタで正確に輪郭の検出を行うことはできない．

たとえば，台形に盛り上がった土地を想像してみると明らかに輪郭が存在するが，隣り合った場所に絶壁のような極端な高低差はない．そこで隣接した画素の輝度差の変化に着目し，上

News Flash――パナソニックコミュニケーション，世界初のIPv6対応カラー・レーザ・プリンタを開発
パナソニックコミュニケーションズ(株)は，世界初となるIPv6対応のカラー・レーザ・プリンタを開発した．また，IPv6 Ready Logo(フェーズ1)を取得した．本年12月に商品化を予定している．

0	1	0
1	－4	1
0	1	0

図８　４近傍ラプラシアン

1	1	1
1	－8	1
1	1	1

図９　８近傍ラプラシアン


写真６　４近傍ラプラシアン


写真７　８近傍ラプラシアン

下，左右，それぞれについて変化の変化（つまり２次微分）の起きる場所を見つけたほうが，より自然に輪郭を得ることができる．

数学表記では，ラプラシアン

$$L=\frac{\partial^2}{\partial i^2}+\frac{\partial^2}{\partial j^2}$$

である．

２次微分とはいえ，最終的には近傍フィルタによる実装可能な単純な差分で表現できる．図８に４近傍ラプラシアンのオペレータを示す．この驚くほど単純な２次微分オペレータは以下の式から導かれる．

水平方向

$$\frac{\partial A}{\partial i}=A(i+1,\ j)-A(i,\ j)$$

$$\frac{\partial^2 A}{\partial i^2}=\frac{\partial}{\partial i}\{A(i+1,\ j)-A(i,\ j)\}$$

$$=\{A(i+1,\ j)-A(i,\ j)\}-\{A(i,\ j)-A(i-1,\ j)\}$$

$$=A(i+1,\ j)-2A(i,\ j)+A(i-1,\ j)\ \cdots\cdots\cdots\cdots(1)$$

垂直方向

$$\frac{\partial A}{\partial j}=A(i,\ j+1)-A(i,\ j)$$

$$\frac{\partial^2 A}{\partial j^2}=\frac{\partial}{\partial j}\{A(i,\ j+1)-A(i,\ j)\}$$

$$=\{A(i,\ j+1)-A(i,\ j)\}-\{A(i,\ j)-A(i,\ j-1)\}$$

$$=A(i,\ j+1)-2A(i,\ j)+A(i,\ j-1)\ \cdots\cdots\cdots\cdots(2)$$

ゆえに４近傍ラプラシアンは，

$$B(i,\ j)=\frac{\partial^2 A}{\partial i^2}+\frac{\partial^2 A}{\partial j^2}$$

$$=A(i-1,\ j)+A(i+1,\ j)-4A(i,\ j)+A(i,\ j-1)$$

$$+A(i,\ j+1)$$

となる．

ここで，２次微分が，ある隣接する部分の差とそれより一つ前の差の差分（差の差）になっているところに注目してほしい．この結果から横方向，縦方向についてこの係数を見ると，結局は隣接する二つの輝度から自分自身の輝度を引いて，自分が周囲とどれくらいかけ離れているかを計算しているということがわかる．

４近傍に斜め方向の検出を加えて改善したものが図９の８近傍ラプラシアン・オペレータである．

４近傍，８近傍ラプラシアン・フィルタの演算結果を写真６と写真７に示すが，明らかに８近傍のほうが検出精度は高い．

ラプラシアン・フィルタの効果を，今回も１次元で確認してみよう．図10のＡを入力として（１，－２，１）の１次元マスクを i＝１～10に摘要してみる．

真ん中の値を－２にしたわけは，足して０にして直流成分を出さないためである．２次元の８近傍の場合は，周囲が８か所なので－８にすれば足して０になる．

i＝１の値は０＋０＋０＝０
i＝２の値は０－０＋５＝５
i＝３の値は０－10＋10＝０
i＝４の値は５－20＋10＝－５
i＝５の値は10－20＋10＝０
i＝６の値は10－20＋10＝０
i＝７の値は10－20＋５＝－５
i＝８の値は10－10＋０＝０
i＝９の値は５－０＋０＝５

これをプロットすると，図10のＢのようになり，傾きが変化する部分のみで値をもつことが確認できる．

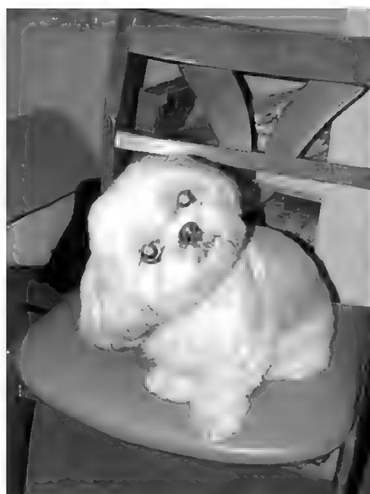
● ハイパス・フィルタ

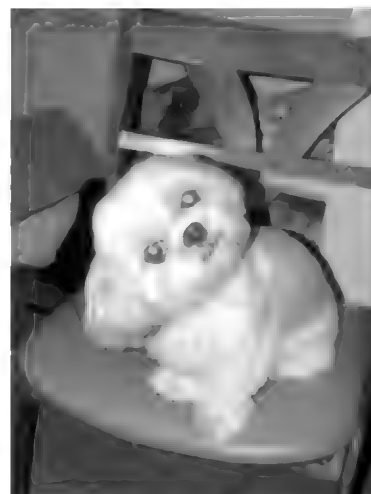
ハイパス・フィルタは，図11の係数をもち，写真８のように輝度変化の大きな部分，つまり細かいディテールの部分を強調して，くっきりさせる作用がある．

前述のエッジ抽出系のフィルタと異なって，中央の係数が周囲の係数の和より絶対値が大きくなっている点に注意してほしい．この中央の係数をさらに大きくしたものが写真９の鮮鋭化フィルタであるが，ハイパス・フィルタに比べて効果がやや柔らかくなっている．中央の係数が大きいということは，相対的に周囲の影響が少ないということなので，これは当然の結果で

New Products――リコー，同社のディジタル複合機やプリンタのためのJavaアプリケーションの開発ツールを開発
（株）リコーは，同社のディジタル複合機やプリンタ上で動作するJava言語のアプリケーションの開発ツール「Embedded Software Architecture」を開発した．現在，パートナ企業にこのツールを公開し，アプリケーションの開発を呼びかけている．


写真8　ハイパス・フィルタ


写真9　鮮鋭化フィルタ


写真10　ガウシアン・フィルタ

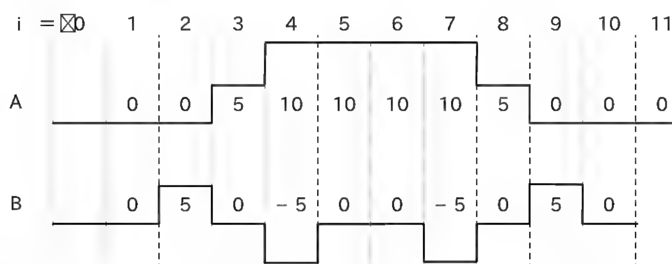


図10　ラプラシアン・フィルタの入力と出力


写真11　斜めSobel

－1	－1	－1
－1	9	－1
－1	－1	－1

図11　ハイパス・フィルタ

1	2	1
2	4	2
1	2	1

図12　ガウシアン・フィルタ

2	1	0
1	0	－1
0	－1	－2

図13　斜めSobel

ある．

● ガウシアン・フィルタ

図12の係数を持つこのフィルタは，1次平滑フィルタより強いローパス効果がある(写真10)．

● 斜めSobel

Sobelフィルタは係数を図13のように斜め方向に配置することにより，斜め方向のエッジ抽出フィルタを作ることができる(写真11)．組み合わせは4方向が考えられるが，サンプルは左上から右下へのケースである．

サンプルのプログラムを章末のURLからダウンロードして，上記のフィルタを実行して効果を確かめることができる．Adobe Photoshopを持っている読者であれば，[フィルタ]→[その他]→[カスタム]で最大5×5のマスク・フィルタに任意の係数を入力して効果を試すことができる．

● PC上でのソフトウェアとしての実装例

では，実際のプログラムを概観してみよう．次のリスト1が3×3マスク・フィルタの実装である．画像は，モノクロ(グレースケール)8ビット画像を想定している．

入力パラメータの`pSrc`は，元画像のバッファへのポインタで，`pDst`は演算結果を返す画像バッファ，`nWidth`と`nHeight`は画像の幅と高さのピクセル数，`pMask`は3×3マスクへのポインタである．図1でいえば，$M11$，$M12$，$M13$，$M21$，$M22$，$M23$，$M31$，$M32$，$M33$の順に整数値が9個並んでいる．

`pMask`は10番目の値として全体の割り算係数をもっている．たとえば，図2の平滑フィルタでは0.1，0.2という係数を1，2の整数で計算するため，最後に10で割っている．上下左右の

Information——イーソル，Flash Player SDKの販売，ポーティング・ビジネスを開始
マクロメディア(株)とイーソル(株)は，Flash Player SDKについてライセンス契約を結び，イーソルが同社の顧客に対してFlash Player SDKの販売，およびポーティング・ビジネスを開始した．

リスト 1
3× 3マスク・フィルタの実装

```
void filter3x3( BYTE* pSrc, BYTE* pDst, int nWidth, int nHeight, int* pMask )
{
    BYTE      *p1, *p2, *p3, *q;
    int       x, y, linewidth, len, sum;

    linewidth = ( nWidth + 3 ) & ~3; ◀────────── ①
    len = linewidth * nHeight;
    memcpy( pDst, pSrc, len );

    for( y = 1; y < nHeight-1; y++ ){
        p1 = pSrc + linewidth * ( y - 1 ) + 1; ◀────────── ②
        p2 = pSrc + linewidth * y + 1; ◀────────── ③
        p3 = pSrc + linewidth * ( y + 1 ) + 1; ◀────────── ④
        q  = pDst + linewidth * y + 1;
        for( x = 1; x < nWidth-1; x++ ){
            sum  =  *(p1-1) * pMask[0] + *p1 * pMask[1] + *(p1+1) * pMask[2] +
                    *(p2-1) * pMask[3] + *p2 * pMask[4] + *(p2+1) * pMask[5] +
                    *(p3-1) * pMask[6] + *p3 * pMask[7] + *(p3+1) * pMask[8];
            if ( pMask[9] != 1 )
                sum /= pMask[9];
            if ( sum > 255 )   sum = 255;
            else if ( sum < 0 ) sum = 0;
            *q++ = (BYTE)sum;
            p1++;     p2++;     p3++; ◀────────── ⑤
        }
    }
}
```



図 14　ソフトウェアの画面

端のライン，カラムを除く内側を演算対象とするので，あらかじめ元画像をコピーしておく．

ここでは簡略化のためにフルコピーを行っているが，画像が大きい場合は必要な部分だけコピーするようにしたほうがよいだろう．

①では水平1行分のバイト数を計算している．これはビットマップ・ファイルの規格上，1行のバイト数が4の倍数でなければならないためで，幅が4で割り切れない場合は0がパディングされて nWidth と一致しないからである．

②，③，④で p1，p2，p3 にピクセル (x, y-1)，(x, y)，(x, y+1) のポインタをセットして，その近傍のピクセルにアクセスしている．

ビットマップ形式では，ヘッダの biHeight が正の数値なら，バッファ先頭が左下ピクセルに相当し，ボトム・アップで数えるようになっているので，マスクの係数が非対称の場合には注意が必要である．

⑤でポインタを一つ右のピクセルに進めて，同様の演算を1行分にわたって行う．全体のアプリケーションは SimpleFilter という Windows プロジェクトを VC++ で作成している．

図 14のように［画像読み込み］ボタンでモノクロのビットマップ・ファイルを読み込んで，フィルタの種類をラジオ・ボタンで選択し，［フィルタ処理］ボタンをクリックすると，左下に3× 3のマスクの値を表示して，フィルタ演算結果の画像を並べて表示するようになっている．

なお，画像のサイズが 256× 256を超えた場合は，左上部分しか表示されないので，注意してほしい．それから，計算は全ピクセルで行っている．

［画像保存］ボタンをクリックすれば，演算結果を別のビットマップ・ファイルとして保存できる．

3 フーリエ変換のイメージをつかむ

直交変換とは「見える世界」から「見えない世界」への変換である，という定義を耳にしたことがあるが，音声や画像のフーリエ変換においては，「見えない世界」は周波数領域というあの世（？）に相当する．

直交変換を行うことで，元の世界（時間領域）から周波数領域に変換して問題（たとえば微分方程式）を解き，その答を逆変換で元の世界に戻す，というような使い方がされる．現世では解決困難な問題を「あの世」で解決したうえ，この世に戻して活用するというイメージであろうか．

● フーリエ級数

まずは簡単な1次元の波形，たとえば音声波形について考えてみよう．

いちばん単純な振動波形は，正弦波（サイン波）である．図 15のように回転する円盤が軸方向に移動するとき，円盤に張り付いた点 A が描く軌跡を横から眺めたものと考えればわか

Information ――フュージョン・コミュニケーションズとパワードコム，7月1日付けで電話事業を統合
フュージョン・コミュニケーションズ（株）と（株）パワードコムは，7月1日付けで両社の電話事業を統合した．これにともない，パワードコムが提供していた東京電話などの電話サービスは，フュージョン・コミュニケーションズのサービスとして提供していく．

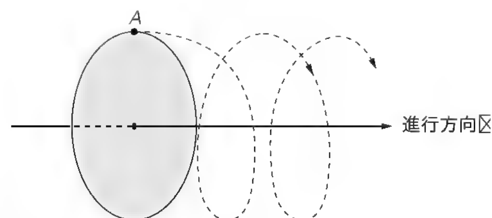


図15　正弦波のイメージ

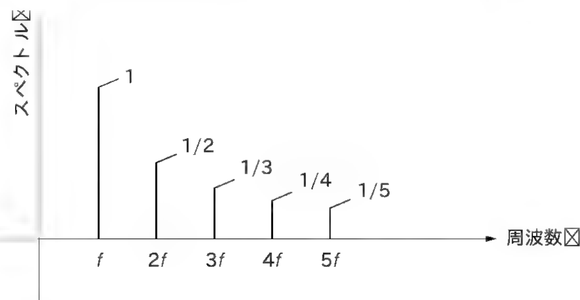


図16　のこぎり波のスペクトル

りやすい．コイル状のバネを引き延ばした形である．

正弦波と余弦(コサイン)波の位相が90度ずれているのは，単にこの軌跡を横から眺めるか上から眺めるかという視点のずれを表しているにすぎない．

音声の場合は横軸が時間で縦軸が振幅を表す．

任意の周期波形は，基本周期の整数倍の周波数をもつ正弦波および余弦波形(高調波または倍音という)の組み合わせ(加算)で表現できる，というのがフーリエ級数である．

$$f(t)=\frac{a_0}{2}+\sum_{n=1}^{\infty}\left\{a_n\cdot\cos(n\omega_0 t)+b_n\cdot\sin(n\omega_0 t)\right\}$$

たとえば，基本正弦波に $1/N$ の大きさをもつ周波数 N 倍の正弦波をどんどん重ねていくと，図16のようにのこぎり波に近づいていく．N が奇数(1，3，5，7…)のみの場合は，矩形波になる．のこぎり波の周波数スペクトルは図16のように表すことができる．

f は基本周波数で，$2f$ は2倍音，$3f$ は3倍音…の高調波成分である．基本波から3倍音までを順に加算してのこぎり波に近づいていくようすを図17に示す．

● フーリエ変換

この任意の時間信号を周波数領域で表すための変換がフーリエ変換である．

具体的には，フーリエ変換は前述のフーリエ級数の式で係数 a，b を求める作業になる．

入力信号 $f(t)$ にフィルタ $g(t)$ を作用させて出力信号 $h(t)$ を作り出す場合，

$$h(t)=\int f(t-x)g(x)dx$$

という定積分が行われるが，これをコンボルーション(畳み込み)という．

上の式を見てわかるように，ある時間 t におけるフィルタ出力は，少し(x だけ)過去の入力値にフィルタ定数 $g(x)$ をかけ算したものを一定の範囲で積算した値である．

この f，g，h をそれぞれフーリエ変換したものを F，G，H とすれば，周波数 s 領域では，

$$H(s)=F(s)\cdot G(s)$$

となる．つまり，元の時間領域での畳み込み演算が周波数領域では単なるかけ算になるのである．

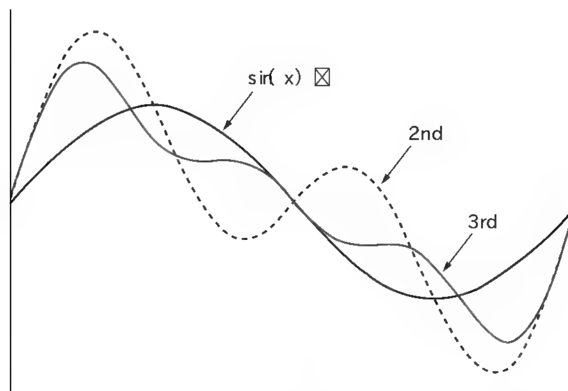


図17　基本波から3倍音までを順に加算してのこぎり波に近づいていくようす

ω

図18　正弦波のフーリエ変換

フーリエ変換の定義式は，

$$F(\omega)=\int_{-\infty}^{\infty}f(t)\cdot e^{-j\omega t}dt \quad (j\text{は虚数単位 } j^2=-1)$$

である．オイラーの公式，

$$e^{\pm j\omega t}=\cos\omega t\pm j\sin\omega t$$

から，$e^{-j\omega t}$ は複素平面上の単位円であるが，わかりやすく言うと複素領域での正弦波である．

したがって，上記のフーリエ変換式は元の波形 $f(t)$ に正弦波を掛け合わせて相関をとっている，と見ることができる．

相関とは，二つの信号がどの程度似ているか，ということを表す数値である．二つの波形がピッタリ同じ場合，その相関は最大になる．つまり，フーリエ変換はあらゆる周波数についてその周波数の正弦波および余弦波と元の波形がどれだけ似ているかということを調べて，周波数軸にプロットしていくプロセスなのである．

Information ——フラッシュ・メモリのベンダであるM-Systems社がACCESSのパートナ・プログラムに参加
米M-Systems社が，(株)アクセスのパートナ・プログラムに参加した．本プログラムでは，ACCESSがツールや情報，サービスを提供している．両社は，RTOSのμMoreで動作するM-Systems社のMobile DiskOnChipを利用するためのサポートを共同で行っていく．

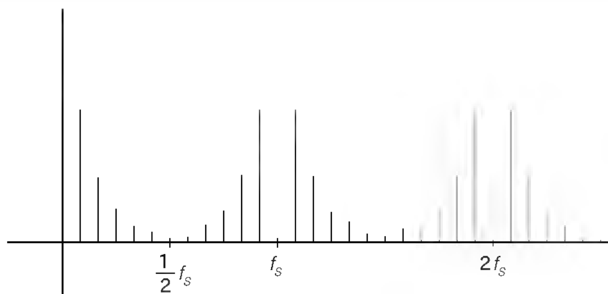


図19　サンプリング

リスト 2　1次元のFFTルーチン

```
void FFT1( int nSize, int nExp, double* pReal, double* pImag,
double *pSinTbl, double *pCosTbl )
{
    int       i, j, k, m, n, len, ph, id1, id2, odd;
    double    re1, re2, im1, im2, c;
    double    itemp[_MAX_SIZE], rtemp[_MAX_SIZE];

    n = 1;
    len = nSize;
    for( i = 0; i < nExp; i++ ){
         len >>= 1;
         m = 0;
         for( j = 0; j < n; j++ ){
              ph = 0;
              for( k = 0; k < len; k++ ){     ←―――――― ①
                   /*       バタフライ演算        */
                   id1 = m + k;
                   id2 = id1 + len;
                   re1 = pReal[id1];
                   im1 = pImag[id1];
                   re2 = pReal[id2];
                   im2 = pImag[id2];
                   pReal[id1] = re1 + re2;
                   pImag[id1] = im1 + im2;
                   re1 -= re2;
                   im1 -= im2;
                   pReal[id2] = re1 * pCosTbl[ph]
                              - im1 * pSinTbl[ph];
                   pImag[id2] = re1 * pSinTbl[ph]
                              + im1 * pCosTbl[ph];
                   ph += n;
              }
              m += ( len * 2 );
         }
         n <<= 1;
    }
    /*      Bit Reversal       */
    for( i = 0; i < nSize; i++ ){     ←―――――― ②
         n = 0;
         odd = i;
         for( j = 0; j < nExp; j++ ){
              n |= ( odd & 0x01 );
              n <<= 1;
              odd >>= 1;
         }
         n >>= 1;
         rtemp[i] = pReal[n];
         itemp[i] = pImag[n];
    }
    for( i = 0; i < nSize; i++ ){
         pReal[i] = rtemp[i];
         pImag[i] = itemp[i];
    }
    /*    ノーマライズ    */
    c = sqrt((double)nSize );
    for( i = 0; i < nSize; i++ ){
         pReal[i] /= c;
         pImag[i] /= c;
    }

}
```

純粋な正弦波をフーリエ変換すると，図18のように周波数ωのところに一本の線が得られ，当たり前の話であるが，周波数ωの正弦波とのみ相関を持つ波形で元の信号が構成されていることを示している．

また，$t=0$においてのみ値を持つ理想インパルスのフーリエ変換は，

$$F(\omega)=\int_{-\infty}^{\infty}1\cdot e^{-j\omega_0}dt=1$$

のようにすべての周波数にわたって一定値，つまり，あらゆる周波数を同等に含んだ波形となる．周波数領域でのフィルタ応答$H(s)=F(s)\cdot G(s)$を考えると，時間領域でのフィルタの畳み込み係数$g(x)$はそのフィルタのインパルス応答(インパルスを入力してフィルタから出てくる波形)であることがわかる．

インパルス応答がわかれば，それをフーリエ変換することで複素周波数特性がわかり，逆に周波数，位相特性が決まればそれを逆フーリエ変換である，

$$f(t)=\frac{1}{2\pi}\int_{-\infty}^{\infty}F(\omega)\cdot e^{-j\omega t}d\omega$$

よりインパルス応答$g(t)$を求めることができ，畳み込みフィルタを時間軸で実現できるのである．

● DFTとFFT

以上は信号がアナログの連続量を扱う場合の話であった．ディジタル信号処理においては，元の入力信号を数値化するために一定の時間間隔でサンプリング(標本化)が行われる．

ほかの章でも述べられているので詳細は省くが，このサンプリングによって周波数特性は0からサンプリング周波数f_sを折り返しながら無限に繰り返していく性質を持つ(図19)．

フーリエ変換は，離散フーリエ変換(DFT：Discrete Fourier Transform)となり，周期がNのサンプル列のDFTは，

$$F(k)=\sum_{n=0}^{N-1}f(n)\cdot e^{\frac{-jk2\pi n}{N}}$$

のように積分の代わりにΣによって表される．

● FFT

高速フーリエ変換(FFT：Fast Fourier Transform)は，離散フーリエ変換の係数の対称性に着目してその演算量を減らし，高速に変換を行う手法である．

周期Nの離散フーリエ変換では，複素数の乗算をN^2回行う必要があるが，高速フーリエ変換ではその乗算回数を$2N\log_2 N$回に減らすことができる．

ここでは具体的なアルゴリズムを解説する代わりにリスト2に1次元のFFTルーチンを掲載する．

関数の引き数は，`nSize`がサンプルのサイズで，これは2のN乗でなければならない．`nExp`はそのNである．`pReal`は実数部入力データで，0, …, 1.0の範囲にノーマライズされている．虚数部`pImag`はすべて0にしておく．`PSinTbl`，`pCosTbl`はサインとコサインのテーブルで，ちょうどサンプル数で2πになる値を設定しておく．

表1 nの値の変化

i	$n(j=0, 1, 2)$
0	0 0 0
1	1, 2, 4
2	0, 1, 2
3	1, 3, 6
4	0, 0, 1
5	1, 2, 5
6	0, 1, 3
7	1, 3, 7

演算の結果は`pReal`，`pImag`にそれぞれ実数部，虚数部が返される．一番内側のループの①がバタフライ演算と呼ばれる部分である．この演算によって入れ替わった順序を元に戻す作業が②のビット・リバーサルである．

$N=3$のサンプル数8の場合を例にとれば，jが0から2に進むにつれて，nの値は表1のように変化する．

● 二次元 FFT

2次元のDFTは，画像の横方向，縦方向について1次元DFTを繰り返し実行することで得られる(図20)．

2次元DFTを式で表すと，

$$F(X, Y)=\frac{1}{M \cdot N}\sum_{y=0}^{N-1}\sum_{x=0}^{M-1} f(x, y) W_1^{X \cdot x} \cdot W_2^{Y \cdot y}$$

$$W_1^{X \cdot x}=e^{-j \cdot \frac{2\pi Xx}{M}}, \quad W_2^{Y \cdot y}=e^{-j \cdot \frac{2\pi Yy}{N}}$$

となる．

DFTは高速化のためにFFTで演算するが，上式のW(回転子と呼ぶ)の行列の対称性を利用して計算量を減らすのである．

リスト3(p.86)に2次元FFTのリストを示す．関数の引き数の`nSize`は，画像の縦横サイズ(2のN乗で同じとする)，`pImage`は画像イメージ・バッファ，`pResult`は演算結果を受け取るバッファへのポインタで，バッファのサイズは画像イメージ・バッファと同じである．

まず，演算用のバッファを実数部，虚数部について2系統確保し，サイン，コサインのテーブルを作成したあと，①で0, 1, …, 255の値を持つグレー・スケールの画像データを`pImage`から演算バッファの実数部に読みこみ，②で水平方向のFFTをすべての行について実行する．

次に，③でこの結果を二つ目の演算バッファに行と列を入れ替えながらコピーする．二つ目のバッファに対しても，各行についてFFTを実行する(④)．行と列を入れ替えているので(つまり90度回転している)，今度は縦方向のFFTを行ったことになる．

こうして得られた2次元FFTの結果は図21のように外側が低周波，中央が一番高い周波数成分となるので，見やすいように中央が直流成分，外側が高周波成分となるように⑤で並び変えている(図22)．

そして⑥で，実数部の2乗と虚数部の2乗を足して絶対値を求め，それの対数をとってパワー・スペクトルを求めている．

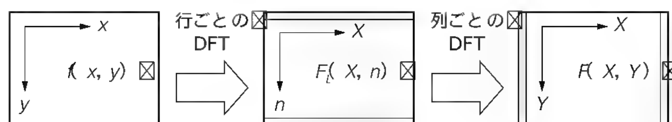


図20 2次元のDFTのイメージ

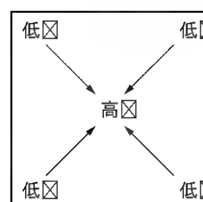


図21 2次元FFTの結果

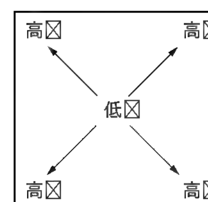


図22 並び変えた2次元FFTの結果

最後の結果は0, …, 255の整数レンジに戻して，`pResult`バッファにコピーする(⑦)．

4 パターン認識の方法

画像を特定の形状と認識したり，画像の中から特定の対象物を抽出するのが画像におけるパターン認識である．

人間が図形を認識するのは，コラムと呼ばれる大脳皮質にある小さな円柱状の組織で三角形や丸といった単純形状を認識しているらしい．コンピュータにおいても，ものを「三角形である」と認識するためには，前段階として，まずは境界の線を抽出する必要がある．次にその線の特徴的な部分(三角形の場合は三つの角)を抽出することで，それが三角形であると初めて判断できる．

あらかじめ学習，または準備された複数の標本のうち，抽出した特徴パラメータがどれにどれだけ近いかを計算し，一番近いものを選び出すことで特定の形状と認知することができる．

通常その「近さ」は，空間距離(ユークリッド距離)で計算される．パラメータの数をNとして，標本をP_i，未知入力をP_jとすると，空間距離は，

$$D=\sqrt{\sum_{k=1}^{N}\left(P_{i,k}-P_{j,k}\right)^2}$$

である．

本章では，2次元FFTの応用として簡単な図形認識を行ってみる．プログラムとソース・コードは章末に掲載したURLからダウンロードできる．

これはあくまでもフーリエ変換の性質を理解するためのサンプルであるので，これだけでは実用にならないことをお断りしておく．本サンプル・プログラムでは，入力画像の周波数成分だけを図形認識のための特徴パラメータとして使っているが，

リスト 3　2次元FFT

```
int FFT2( int nSize, unsigned char *pImage, unsigned char
                                                    *pResult )
{
    int       i, j, n, exp, h;
    double    *re[_MAX_SIZE];
    double    *im[_MAX_SIZE];
    double    *re2[_MAX_SIZE];
    double    *im2[_MAX_SIZE];
    double    max, abs, co, x, y, db;
    double    *sintbl, *costbl;
    unsigned char *p;

    /*    サイズのチェック    */
    if ( nSize > _MAX_SIZE )
        return    -1;

    n = 1;
    for( exp = 1; exp <= _MAX_EXP+1; exp++ ){
        n <<= 1;
        if ( n == nSize )
            break;
    }
    if ( exp == _MAX_EXP+1 )
        return    -2;

    /*        演算バッファの確保        */
    for( i = 0; i < nSize; i++ ){
        re[i]  = (double *)malloc( nSize * sizeof(double));
        im[i]  = (double *)malloc( nSize * sizeof(double));
        re2[i] = (double *)malloc( nSize * sizeof(double));
        im2[i] = (double *)malloc( nSize * sizeof(double));
    }

    /*        サイン,コサイン・テーブルの作成        */
    sintbl = (double*)malloc( nSize * sizeof(double) );
    costbl = (double*)malloc( nSize * sizeof(double) );
    co = -2.0 * _PAI / nSize;
    for( i = 0; i < nSize; i++ ){
        sintbl[i] = sin( co * i );
        costbl[i] = cos( co * i );
    }

    /*        画像を配列に取り込む    */
    for( j = 0; j < nSize; j++ ){              ←①
        for( i = 0; i < nSize; i++ ){
            re[i][j] = (double)*( pImage + nSize * j + i )
                                                    / 255.0;
            im[i][j] = 0.0f;
        }
    }
    /*        水平方向 FFT        */
    for( i = 0; i < nSize; i++ ){              ←②
        FFT1( nSize, exp, re[i], im[i], sintbl, costbl );
    }
    /*        配列の転置            */
    for( i = 0; i < nSize; i++ ){              ←③
        for( j = 0; j < nSize; j++ ){
            re2[j][i] = re[i][j];
            im2[j][i] = im[i][j];
        }
    }
    /*        垂直方向 FFT        */
    for( i = 0; i < nSize; i++ ){              ←④
        FFT1( nSize, exp, re2[i], im2[i], sintbl, costbl );
    }
    /*                */
    h = nSize / 2;
    for( i = 0; i < h; i++ ){                  ←⑤
        for( j = 0; j < h; j++ ){
            re[j][i] = re2[j+h][i+h];
            im[j][i] = im2[j+h][i+h];
            re[j+h][i+h] = re2[j][i];
            im[j+h][i+h] = im2[j][i];
        }
        for( j = h; j < nSize; j++ ){
            re[j][i] = re2[j-h][i+h];
            im[j][i] = im2[j-h][i+h];
            re[j-h][i+h] = re2[j][i];
            im[j-h][i+h] = im2[j][i];
        }
    }
    /*        パワー・スペクトル    */
    max = 0.0;
    for( i = 0; i < nSize; i++ ){              ←⑥
        for( j = 0; j < nSize; j++ ){
            x = re[i][j];
            y = im[i][j];
            abs = re[i][j] * re[i][j] + im[i][j] * im[i][j];
            if ( abs == 0.0 )
                db = 0.0;
            else
                db = 10.0 * log( abs ) + 80.0;
            re[i][j] = db;
            if ( db < 0.0 )
                db = 0.0;
            else if ( db > max )
                max = db;
        }
    }
    p = pResult;
    for( i = 0; i < nSize; i++ ){              ←⑦
        for( j = 0; j < nSize; j++ ){
            *p++ = (unsigned char)( 255.0f * re[i][j] / max );
        }
    }
    /*        バッファとテーブルの解放        */
    for( i = 0; i < nSize; i++ ){
        free( re[i] );
        free( im[i] );
        free( re2[i] );
        free( im2[i] );
    }
    free( sintbl );
    free( costbl );

    return    0;
}
```

実用的なプログラムでは入力画像のノイズを除去したあと細線化処理を経て抽出した線分をベクトル化する方法がよく用いられる．

また，フーリエ変換では元の時間軸情報(位置情報)は消えてしまうので，このような目的にはウェーブレット変換による多重解像度解析のほうが適していると思われる．

まずは2次元FFTプログラムを実行してみよう．

[Open]ボタンでビットマップ・ファイルを読み込むと，画像が左上に表示される．続けて[FFT]ボタンを押すと，2次元FFTを実行し，右側にその結果を表示する．

写真12～写真17に元の画像とそのFFTの結果を示す．左が元の画像で，右側が得られたスペクトルである．スペクトルの真ん中がDC成分で，両端に行くほど高い周波数領域を表している．縦，横の線で構成される画像は，スペクトルも縦と横の成分を持っていることがわかる．

チェス盤模様のスペクトルが細かい構造を持っているのは高調波成分を多く持っているからである(写真12)．チェス盤模様は，横方向に注目すると黒と白が交互に繰り返されているので矩形波である．そのため，奇数N次の高調波が$1/N$で含まれていると考えられる．

小さな白い正方形のフーリエ変換は，想像より複雑なスペクトルの形をしている．元の図形は正方形が1個だけなので，イ

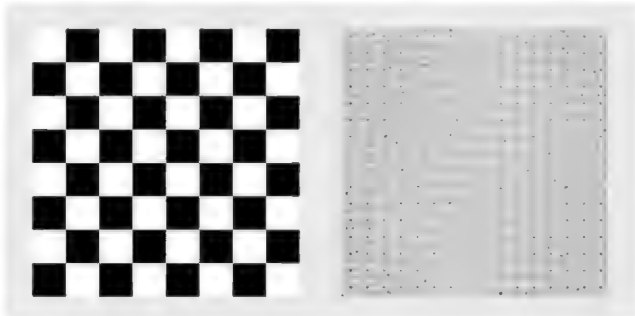


写真12　2次元FFTの結果(1)

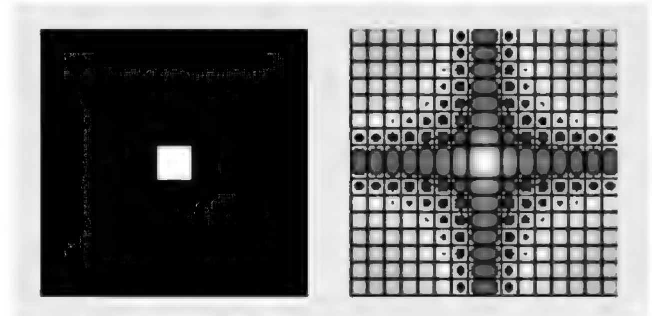


写真13　2次元FFTの結果(2)

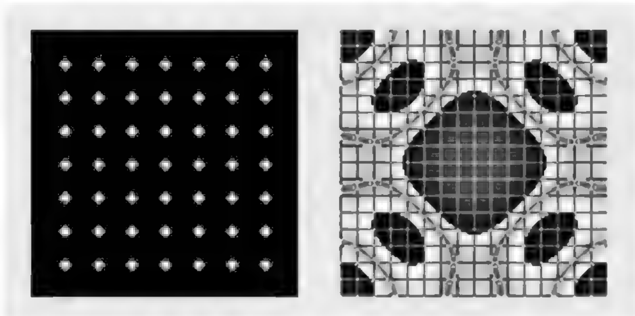


写真14　2次元FFTの結果(3)

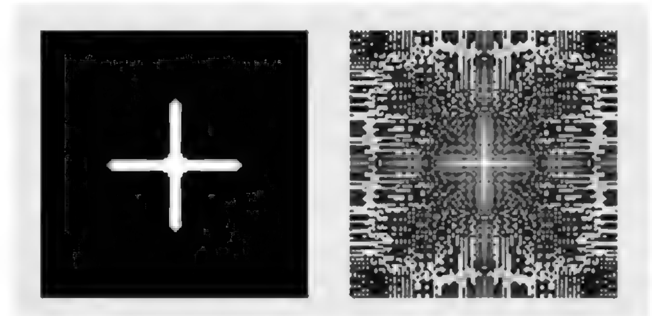


写真15　2次元FFTの結果果(4)

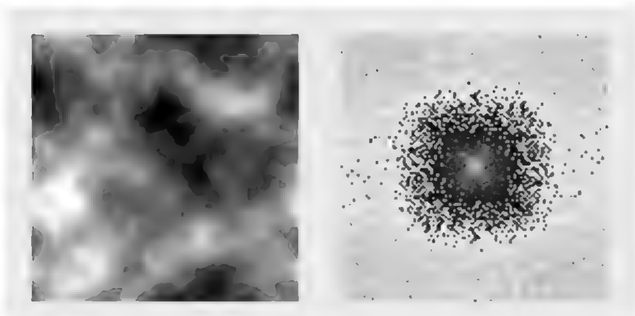


写真16　2次元FFTの結果(5)

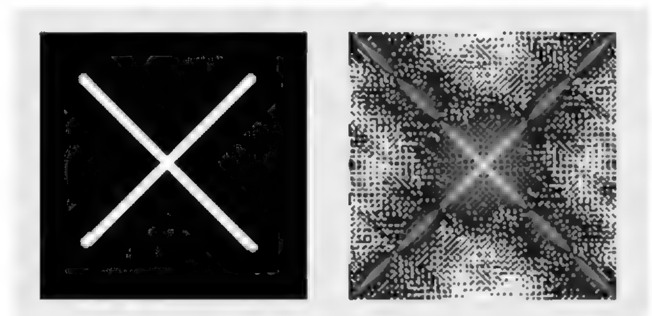


写真17　2次元FFTの結果(6)

ンパルスに近い波形になるわけで，全周波数帯域にわたってスペクトルが存在していることがうなずける．四角形の幅(基本周波数)の整数倍のところでスペクトル値が低く(暗く)なって，一種の櫛形フィルタの形になっていることがわかる(写真13)．

小さなダイアモンドが8×8個ほどメッシュ状に並んだ図形(写真14)では，興味深い結果が得られた．8×8の規則性を表す格子状の線に加えて，ダイヤモンドの持つ斜め方向の成分がフラクタル図形のようなおもしろい模様を作っている．[中心部拡大]のラジオ・ボタンを押すと，低周波領域を拡大して見ることができる．

さて，このプログラムを使って簡単な図形認識を行ってみよう．[Clear]ボタンで画像を消去して，マウス・カーソルを左の元画像の領域に持っていくと，カーソルがペンの形に変わってフリー・ハンドで描画が可能になる．

たとえば，これで適当な図形を描いて[FFT]ボタンを押すと，その図形のスペクトルが右側に表示されると同時に，○，△，×，□の四つの図形の，あらかじめ計算してあるテンプレートと比較して，近似度をバー・グラフで表示する，という内容である．

特徴パラメータとしては，FFTで計算したスペクトルから中央付近の低周波部分を抜き出して，16×16のスロットに分割して，スロット内の平均値をとっているだけである．特徴パラメータの数が入力サンプルと標本で異なる場合は，DPマッチングなどの手法が使われるが，ここでは特徴パラメータの数は固定なので不要である．

先にも書いたように，フーリエ変換のパワー・スペクトルからわかるのは周波数の特徴だけである．したがって数字の「6」と「9」，ひらかなの「あ」と「お」などは周波数成分がよく似てい

COLUMN

ディジタル画像処理“プチ”用語解説

ここでは，ディジタル画像処理でよく使われている用語と，その概念について解説する．

●ピクセルとカラー深度

液晶モニタなどを拡大して見ると赤，緑，青の小さい点がひとかたまりで光っているのが見える．この3色の階調の組み合わせで一つの「ピクセル」の色が決まる．赤（R），緑（G），青（B）各階調（輝度）をいくらの分解能で表すかを，カラー階調の深さ「カラー深度」と呼ぶ．

たとえば，RGB各カラーを8ビットで量子化した場合，RGB各色の階調は0から255までの値をとることが可能で，24ビット・カラー，フルカラー，Trueカラーなどと言う．モノクロ画像では，階調だけ表せばよいので8ビットで256階調の表現が可能である．

16ビット・カラーでは，RGBの各値が5，5，5の5ビットずつのものと，5，6，5ビットのものがあるが，後者は最近ではあまり使われていない．

それから，GIF形式のように8ビットでカラーを表す場合は，インデックス・カラーと呼び，各データはピクセルの輝度を表すのではなく，パレットのエントリ番号を表す．パレットは通常，一つのエントリがRGB24ビットで構成されている．つまり，任意の色を選べるが，使えるのは256種類だけ，ということである．

画像を保持するために必要なデータ量は，画像の（ピクセル数で表した）幅×高さ×カラー深度になる．たとえば，24ビット・カラーのVGAサイズの画像を保持するために必要な容量は，640×480×3=921,600バイト（約1Mバイト）になる．

カラー画像をモノクロ画像に変換する場合，そのままRGBの値を足して3で割ったのでは人間の視覚感度から自然にならないので，通常は $Y=0.299R+0.587G+0.114B$ という式で変換するが，画像処理の内容によってはR，G，Bの単純平均がとられることもある．

●解像度の単位

ピクセルの数が多いほど画像の解像度は高くなる．写真などの画像の解像度の単位はppi（pixels per inch）がよく使われ，1インチ（25.4mm）あたりにピクセルが何個あるかを表す．プリンタなどのデバイスでは，デバイス解像度が用いられ，単位はdpi（dots per inch）である．

たとえば，XGAサイズ（横1024×縦768ピクセル）で撮影したデジカメ写真を360 dpiのプリンタで印刷すると，大きさは横方向は1024/360×25.4=72.25mm，縦方向は768/360×25.4=54.19mmのサイズになる．

●ビットマップ

画像をビット（ピクセル）の並びとして表現したものをラスタ画像とか，ビットマップと呼ぶ．ビットマップは，そのまま拡大すると四角いピクセルのギザギザが目立ってしまう．

それに対して，EPS形式のようにベクトルの集まりで表現したものをベクタ画像と呼び，拡大してもどこまでもなめらかで，データ量も抑えることが可能である．しかし，写真のような複雑な画像には向いていない．

描画ツールでは，ビットマップを扱うものを「ペイント系」，ベクタ画像を扱うものを「ドロー系」などと呼んで区別している．

●画像ファイル形式

Windowsでは，ビットマップにヘッダやカラー・テーブル（インデックス・カラーの場合）を付加して拡張子「.bmp」で保存したファイルを標準形式としている．基本は非圧縮だが，モノクロのビットマップ・ファイルではランレングス圧縮が使われることもあるようだ．

デジカメの標準形式であるJPEGは，DCT（離散コサイン変換）を施して周波数成分で並べ替えたあと，頻度の高いデータほど短い符号を割り当てることで（ハフマン法）データを圧縮している．このプロセスは，非可逆圧縮なので完全に圧縮前の画像を再現することはできない．

●ガンマ補正

CRTなど，一般の表示装置は入力信号のレベルに対して明るさがリニアに比例していない．入力レベルを V としたとき輝度が V^{γ} のように非線形で変化することをガンマ特性と呼ぶ．ビデオ・カメラなどの入力装置でも，入力と出力信号の関係は非線形である．そのため，入力から出力に至る経路で補正を行わないと，正しい映像は再現されない．これを補正する操作をガンマ補正と呼ぶ．

●ヒストグラムとコントラスト

画像に含まれる特定の輝度をグラフにしたものをヒストグラムという．図Aにおいて，横軸は左から右に行くほど高い輝度を表し，縦方向は画像中でその輝度を持つピクセルの数を表す．全体的に暗い画像のヒストグラムは左のほうに重心があり，明るい画像は右寄

るので，FFTだけで識別するのは困難である．

そのかわり，入力領域のどの部分に図形を描いても同様の結果が得られ，さらにおもしろいことに同じ大きさの○を1個だけ描いても，3個描いてもスペクトルはあまり変わらないのである．また，三角形の三辺を分解して別の場所に置いても元のスペクトルとほとんど同じになる．

以上，画像フィルタ，フーリエ変換とそれを2次元画像に摘要した2次元DFT，それの具体的な実現手段である2次元FFTの性質について駆け足で概観してきた．

誌面の関係で舌足らずな部分が多くなってしまったが，実際にプログラムを触って入力図形とその出力の特徴を体感していただき，ディジタル画像処理への興味を深めていただければ筆者にとって，このうえない喜びである．

かどや・じゅんいち　（株）ゼクー

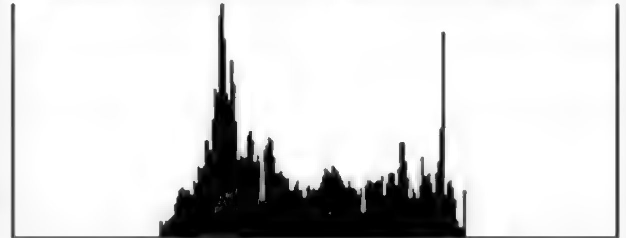
※本章で使用したソフトウェアは，本誌のWebサイトからダウンロードできる．http://www.cqpub.co.jp/interface/



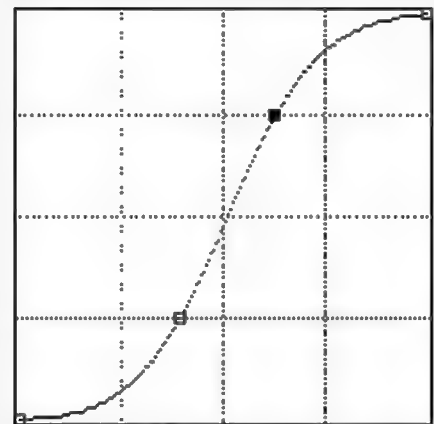


図A　コントラストが強い





図B　コントラストが弱い



図C　トーン・カーブ

りに重心がある．図Bのようなヒストグラムの画像はダイナミック・レンジの一部しか使っておらず，コントラストの悪い画像といえる．反対に図Aの画像はヒストグラムの端から端まですべてのレンジを使っており，コントラストの良い画像である．

● レベル変換

上記のヒストグラムでコントラストの悪い画像の場合，どうやって改善すれば良いのであろうか．図Cのようなカーブでレベルの変換を行えばコントラストを上げてメリハリのある画像に変換することができる．つまり，中間の輝度にエネルギが集まっている状態を，真ん中より明るい部分はより明るく，暗い部分はより暗い状態にして，ヒストグラムの分散を図るのである．

通信分野におけるディジタル信号処理

5 ディジタル変調/復調の基礎と原理

本章では，アナログ方式からディジタル方式への移行が進んでいる通信の分野——その中でも，無線通信の分野におけるディジタル信号処理の基礎について，アナログ分野の考え方を取り入れつつ，その原理を解説する．
(編集部)

長野 昌生

はじめに

通信というと，前章の音声や画像も広い意味で「通信」の範囲に入るのだが，いずれもはじめはアナログ方式で実用化され，その後，性能向上や利便性のために次々とディジタル化が進んでいる分野である．では，アナログ技術はもう不要かというと，そのようなことはない．原理的にも実現技術においても基本はアナログ技術の上に成り立っている．

ひと口に通信といっても，有線による電話通信，無線通信，光通信など，その技術と応用範囲は広いものがある．筆者が携わってきた部分はおもに無線通信の分野である．本章では，アナログ無線通信の基本原理を紹介することにより，ディジタル技術への導入とし，ディジタル技術の本質への理解を深めていただければと考える．

通信におけるディジタル信号処理の世界では，ハードウェアと信号処理が密接につながっており，ソフトウェア的業務に従事する技術者でもハードウェアに対する理解が必要となる．この分野の歴史は，従来はアナログ回路で実現していたものをディジタル信号処理(当然，新規なハードウェアの上)で置き換えることの歴史であり，それは今も続いている．

また，アナログにせよディジタルにせよ，それらの技術の共通の基盤はアルゴリズムであり，それを記述する道具が数学である．数学といっても，その広い世界のほんの一部を学習するだけで信号処理の世界では十分に力を発揮できる(少なくとも最初のうちは)．

ディジタル通信における信号処理といっても，その内容は多岐に渡り，限られたページ内ではとても語りつくせない．そのため本章は，スキーでいうといきなり急斜面から突き落とし，体で覚えてもらうような解説になっているかもしれない．初学者には初めて接する専門用語の連発かもしれないが，最後まで読んでいただければ理解できるように書いたつもりである．多少わからない用語にぶつかっても，がまんして読み進んでいただきたい．

1 電波による情報の伝達

図1は，電波による情報の伝達の概念を描いたものである．左側が送信で，右が受信とした．伝達されるべき情報は古くからある音声のほか，数値データそのものの場合もある．

● アナログ通信方式

アナログ変調方式による場合は，送信側のAの通信内容は変調が直接かけられ，アンテナから放射される(図では省略したが，電波を遠くへ飛ばすためには，アンテナの前に電力増幅器が必要になる)．空中を伝播した電波は受信側のアンテナで捉えられる．多くの場合，その信号は微弱であるので，必要な電力まで増幅器で増幅され，送信側の変調と逆処理の復調処理をされると元の送信信号が復元され，Bの受信信号となる．

情報を電波を媒体として伝達させるためには，少なくとも数kHz以上の搬送波が必要になる．なぜなら，アンテナから電波を放射するには，アンテナのサイズを搬送波の波長と同程度にしなければならないからである．

昨今，携帯電話などで一般化されたディジタル変復調方式においても，システムの概要はアナログ方式と大きくは変わらないが，手続きが複雑になる．まず，送信信号が音声などのアナログ信号の場合はA-D変換器などによりいったん数値化される．では，元々数値化されているディジタル・データはそのまま変調をかければよいかというと，そうではない．変調方式に適合した形式に変換されている．これを図では「フォーマット変換」というブロックで表した．いずれにしても，ディジタル・データはもっとも細かく噛み砕くと0か1かの「ディジタル信号」として扱われる．このようなフォーマット変換された信号を「ベースバンド信号」と呼ぶ．変調方式に応じて1ビット単位であったり，2ビット単位であったり，場合によっては256ビットとやや大きい形で一つのかたまりとして扱われる．このようなベースバンド信号は「変調」により，電波となりや

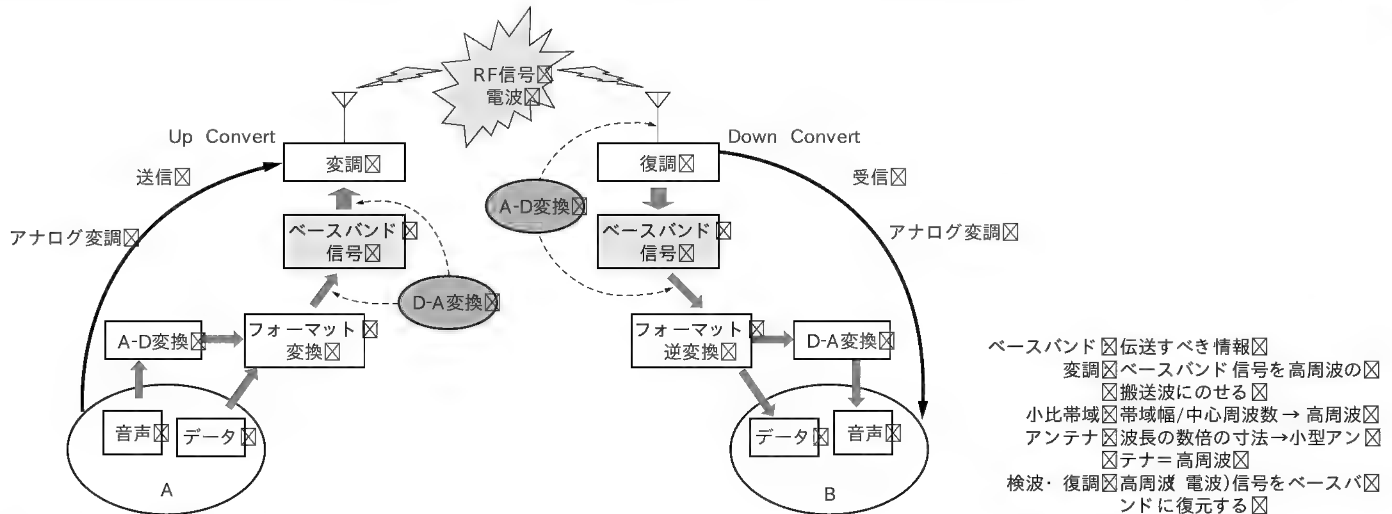


図1　電波による情報の伝搬

すい形式にさらに変換される．これを先ほど述べた「ベースバンド信号」とする．これは，少なくともアンテナまでの段階で，どこかでアナログ信号に戻さなければならない(D-A変換)．ここから先のアンテナから電波が放射される部分は，アナログ方式と同じである．

受信側では，アナログ方式と同じく送信側と逆の手順の処理を行えば，受信側で復元される．復調により，まず「ベースバンド信号」を得るが，これは送信側でのベースバンド信号と同じである．A-D変換をどこで行うかは，変調方式や通信機の設計により異なるので，一概に決めることはできない．一般的には，ベースバンド信号の前後が多い．

A-D，D-A変換をよりアンテナに近いところで行うことを「ソフトウェア無線」，対極のそれを一切行わないものを「アナログ無線」と理解しても，大筋でまちがいはない．

以上の話は，ベテランの方にはわかりきった退屈な話だが，初学者の方には謎めいた単語の羅列でわかりにくい内容に思えるかもしれない．これらの「謎」は以下の内容で徐々に解説していくことにしよう．

● 振幅変調(AM)と周波数変調(FM)

本題のディジタル通信の前に，古典的なアナログ変調(通信)方式について簡単に説明する．ディジタル方式といえども，実機における回路動作の最後はアナログ(電圧と電流の時間変化)である．アンテナからディジタル信号(数字)が直接飛び出るわけではなく，通信機の末端で動く部分は必ずアナログ信号であることを念頭に置いていただきたい．ディジタル通信方式は，見方によってはアナログ方式のちょっと特殊な運用にすぎないとも考えられる．

前置きが長くなったが，図2は代表的なアナログ変調方式である振幅変調(以下AM)の例を時間軸で観測したものである．

式で表すと，

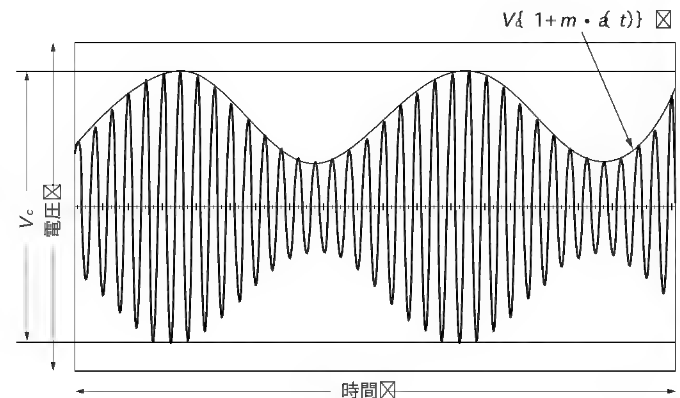


図2　AM変調信号

$$S_{AM}(t)=V_c\{1+m\cdot a(t)\}\cdot\cos(\omega_c t) \quad \cdots\cdots (1)$$

V_cを搬送波の振幅電圧，mを「変調度」，$a(t)$を被伝送信号，ω_cを搬送波の角周波数，tを時間とする．$a(t)$の周波数はω_cに対して十分低いものとし，$|a(t)|\leq 1$とする．

図2では被伝送信号は，$a(t)=\cos(\omega_s t)$と単純な余弦信号とした．式(1)は搬送波周波数ω_cの信号の振幅が，

$$V_c\{1+m\cdot a(t)\} \quad \cdots\cdots (2)$$

で時間変化するという形になっている．振幅の変化は，被伝送信号そのものではなく，オフセットとスケーリングが施されていることに注目してほしい．図2の実線で示した包絡線に対応する．

$$S_{AM}(t)=V_c\times a(t)\times\cos(\omega_c t) \quad \cdots\cdots (3)$$

のように，オフセットをかけない形は通常はとらない．なぜならば，式(3)の左辺では$a(t)$の正負の判別がつかないからである．式(1)のようにオフセットをかけ，式(2)の成分が必ず正

となるようにしなければならない．

一方，図3は周波数（FM）変調信号の例である．式で表すと，

$$S_{FM}(t) = V_c \cos\left(\omega_c t + m \cdot \int a(t)dt\right) \quad \cdots\cdots(4)$$

である．AMと同様に，V_cは搬送波の振幅電圧，ω_cは搬送波の角周波数である．mはAM同様「変調度」と呼ばれる．式（4）の周波数は，cos（ ）のかっこ内が位相である，これを時間で微分したものが周波数であり，

$$\omega_c + m \cdot a(t) \quad \cdots\cdots(5)$$

となる．周波数の変化は，位相の変化の変形であると理解していただきたい．

ここで，被伝達信号$a(t)$は，搬送波の周波数の変化として伝えられている．また，信号の周波数に搬送波というオフセットがかかり，mというスケーリングがかかったものと見ることができる．当然，mが大きければ，信号の周波数範囲は広くなる．

AM変調は「線形変調」とも呼ばれ，変調後の信号の帯域幅が被伝送信号と同一になる．しかしFM変調の場合，式（5）からもわかるとおり，その帯域幅はmに対応して任意に選択できる．一般に信号の帯域幅が広いほど伝送系のS/N比を高くとることができる．

蛇足ではあるが，式（1）や式（4）の変調を実現するアナログ回路が古くから実現されている．AMとFMのアナログ変調に関しては，参考文献（3）に詳しいので，参考にされたい．

2 ディジタル通信の信号形式

● ベースバンド信号

ディジタル通信方式においても，搬送波の位相と振幅の変化により伝達することは同じである．ディジタル変調信号の代表例として，図4（a）に16QAMと，（b）にπ/4DQPSKと呼ばれる方式の「コンスタレーション」（IQダイヤグラムとも呼ぶ）を示した．これは，ベースバンド信号の軌跡を表したものである．水平軸の“I”は同相成分（In Phase）と呼ばれ，Iはその頭文字であり，I成分と呼ばれることもある．同様に垂直軸は直交成分（Quadrature）であり，Q成分である．時間軸は誌面に対して垂直と考えていただきたい．ディジタル無線の世界では，信号を複素数として扱い，I成分を実数部，Q成分を虚数部として扱う．図4の信号は，

$$S(t) = I(t) + jQ(t),\ j\text{は虚数単位},\ j^2 = -1 \quad \cdots\cdots(6)$$

以上のような複素数である信号の時間変化の軌跡を同一面に描いたものである．小さな●は「シンボル点」と呼ばれるもので，一定の時間間隔（シンボル・レート）で，図に描いた所定の位置のどれかを通過するというシステムの約束事になっている．シンボル点だけに信号が存在し，シンボル点からシンボル点は瞬間的に移動できることが理想である．しかし実際には，信号の帯域幅に制限（後述）があるので，信号の軌跡はシンボル点の間を“連続的”に移動することになる．

さて，ここまでで「ベースバンド」や「コンスタレーション」などの謎の用語に面食らった方もいるかもしれないが，おいおい説明していく．

● 振幅と位相を同時に伝達するには

ディジタル変調の場合，その多くは振幅と位相の両方の変化で情報を伝えている．振幅と位相が時間変化する関数として，

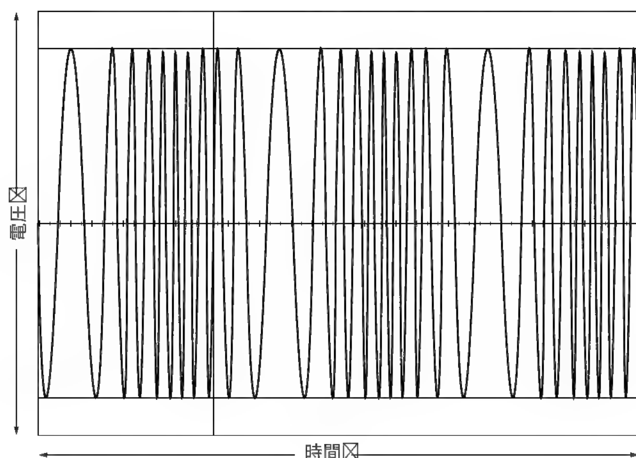
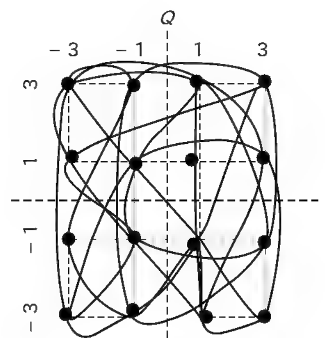
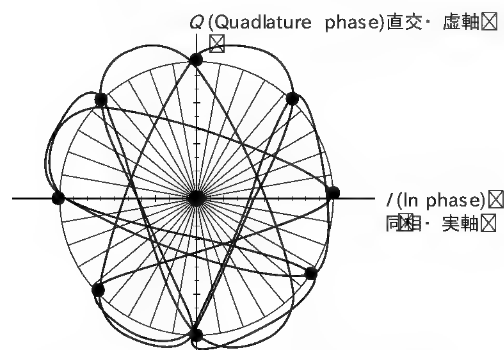


図3　FM変調信号



（a）16QAM信号のコンスタレーション



（b）π/4 DQPSK信号のコンスタレーション

図4　コンスタレーション

それぞれ $A(t)$ と $\theta(t)$ とすると，次の $S(t)$ のように表現できる．

$$S(t) = A(t) \cdot \cos(\theta(t)) \quad \cdots\cdots (7)$$

しかし，信号として観測されるのは $S(t)$ である．図5にその例を示した．式(7)の信号において，

$$\theta(t) = \omega t \quad \cdots\cdots (8)$$

と，式(6)が一定周波数の信号であれば，

$$\theta(t) = \cos^{-1}(S(t) / A(t)) \quad \cdots\cdots (9)$$

のように位相を求めることは可能であるが，これは特殊な場合である．図5のような信号には，式(9)の適用は不適当である．信号の変化が振幅によるものなのか，位相によるものなのか，判別ができないからである．同様に式(7)の振幅を求めることもできない．

AM変調とFM変調の場合，振幅と位相のどちらか一方しか変化させていないので，信号の状態を特定できた．簡単なアナログ回路で実現できたのも，この一つのパラメータしか変化しない，という前提があるからだ．

被伝達信号の振幅と位相だが，複素数を使って表現すると明解になる．図4の信号が式(6)で表現されているとすれば，その振幅 $A(t)$ と位相 $\theta(t)$ は，

$$A(t) = \sqrt{I^2(t) + Q^2(t)} \quad \cdots\cdots (10)$$

$$\theta(t) = \tan^{-1}(Q(t) / I(t)) \quad \cdots\cdots (11)$$

である．ここでの I と Q が式(6)に対応することを改めて強調しておく．

時系列で振幅と位相を扱うためには，このように複素数である2チャネルの信号でなければならないのだろうか．だとすると，いささか不便な話だが，実際にはそうではない．

実際に空中を電波として伝播する信号は，式(6)に搬送波周波数 ω_c を乗じたものの実数部が使われており，次のような形になる．

$$S_{c_RE}(t) = \mathrm{Re}\left[(I(t) + jQ(t)) \times \exp[j(\omega_c t + \theta_0)]\right] \quad \cdots (12)$$

$$= I(t) \cdot \cos(\omega_c t + \theta_0) - Q(t) \cdot \sin(\omega_c t + \theta_0) \quad \cdots (13)$$

j は虚数単位．θ_0 は初期位相である．参考までにこの虚数部は，

$$S_{c_Im} = I(t) \cdot \sin(\omega_c t + \theta_0) + Q(t) \cdot \cos(\omega_c t + \theta_0) \quad \cdots (14)$$

だが，実数部か虚数部かということはあまり重要ではない．

式(13)または(14)の形の信号から ω_c の要素を除去すれば，$I(t)$ と $Q(t)$ の成分だけを取り出し，式(10)と式(11)の情報を導き出せることになる．実際にそれは可能である．

● ディジタル変調信号の送受信

図6は，ディジタル変調方式による送信と受信の概念を描いたものである．

左端の I と Q は，式(6)の信号である，図4のような信号の I 成分と Q 成分が送り込まれているものと理解していただきたい．その右にある点線の四角で示されているものはD-A変換器であり，ここでディジタル信号からアナログ信号に変換される．ただし，D-A変換器が置かれる場所は図中の①②③いずれの位置でもよいし，極端な話，I と Q は元からアナログであってもよい．次に信号は，周波数 ω_c を発生する局部発信器(local)の信号と乗算処理が行われる，ただし I と Q に乗ぜられる信号は，位相を90度ずらし，それぞれ，$\cos(\omega_c t)$ と $\sin(\omega_c t)$ である．加えて I-Q の処理がなされてアンテナに到達する信号は式(13)となる．図6では $S_{RF}(t)$ と示した．

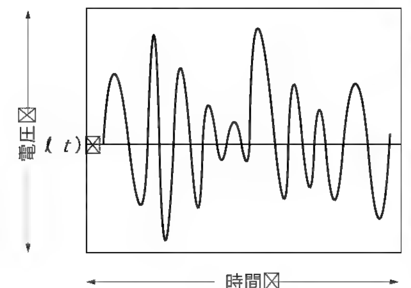


図5
振幅と位相の変化する実数信号

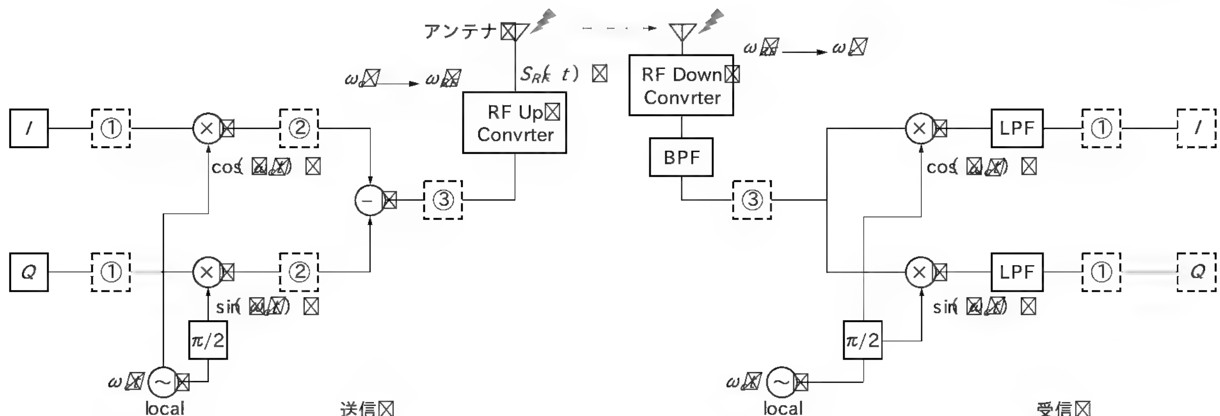


図6　ディジタル変調信号の送受信

$$S_{RF}(t) = I(t) \cdot \cos(\omega_c t + \theta_0) - Q(t) \cdot \sin(\omega_c t + \theta_0) \quad \cdots\cdots (15)$$

受信側では，アンテナから受信した信号は①か③のどちらかでA-D変換される，その場所は送信と同様にさまざまである．

ともあれ，受信側での当初の信号は$S_{RF}(t)$である．これに送信側と同様に局部発信器による，$\cos(\omega_c t)$と$\sin(\omega_c t)$を乗じると，

$$I\text{側}: I_0(t) = \{I(t) \cdot \cos(\omega_c t + \theta_0) - Q(t) \cdot \sin(\omega_c t + \theta_0)\} \times \cos(\omega_c t) \quad \cdots\cdots (16)$$

$$Q\text{側}: Q_0(t) = \{I(t) \cdot \cos(\omega_c t + \theta_0) - Q(t) \cdot \sin(\omega_c t + \theta_0)\} \times \{-\sin(\omega_c t)\} \quad \cdots\cdots (17)$$

$\cos(\ )$と$\sin(\ )$の乗算は，三角関数の和と差の公式を適用して，

$$I_0(t) = 1/2\left[I(t) \cdot \{\cos(2\omega_c t + \theta_0) + \cos(\theta_0)\} - Q(t) \cdot \{\sin(2\omega_c t + \theta_0) + \sin(\theta_0)\}\right] \quad \cdots\cdots (18)$$

$$Q_0(t) = 1/2\left[I(t) \cdot \{-\sin(2\omega_c t + \theta_0) + \sin(\theta_0)\} - Q(t) \cdot \{\cos(2\omega_c t + \theta_0) - \cos(\theta_0)\}\right] \quad \cdots\cdots (19)$$

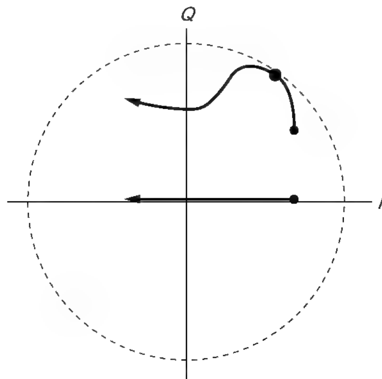


図7
複素信号の実軸への投影

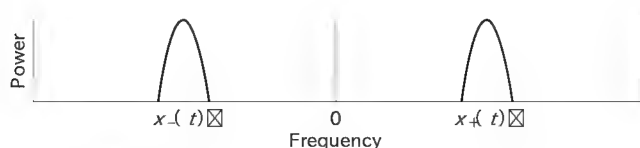


図8 実数信号のスペクトル

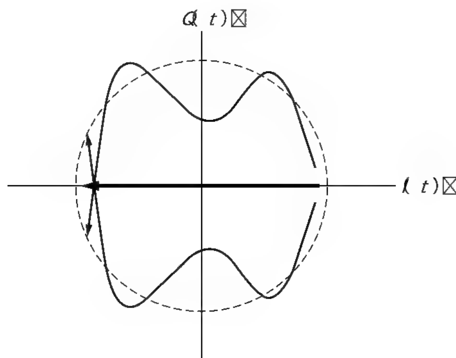


図9
共役な複素信号の
IvsQダイヤグラム

となる．周波数が$2\omega_c$と，周波数を持たない(0Hz)成分の二つが存在する．$2\omega_c$の成分は，低域通過フィルタ(LPF)で除去することができ．その出力は，

$$I_1(t) = 1/2\{I(t) \cdot \cos(\theta_0) - Q(t) \cdot \sin(\theta_0)\} \quad \cdots\cdots (20)$$

$$Q_1(t) = 1/2\{I(t) \cdot \sin(\theta_0) + Q(t) \cdot \cos(\theta_0)\} \quad \cdots\cdots (21)$$

となる．これは，

$$I_1(t) + jQ_1(t) = 1/2\{I(t) + jQ(t)\} \times \exp[j\theta_0] \quad \cdots\cdots (22)$$

の実数部と虚数部と同じである．つまり，送信信号の式(6)がθ_0だけ位相回転したものということになる．式(22)に何らかの補正で$\exp[-j\theta_0]$をかけてやれば，式(22)は式(6)と比べて振幅が1/2であること以外はすっかり同じになる．

大まかに述べると，以上のようなプロセスで，$I+jQ$の複素数の信号を電波にのせて受信側で復元することが可能であることを証明した．

送信側で2チャネルの信号が，途中で1チャネルの電波となり，受信側で再び2チャネルになる．一見，摩訶不思議なことのように思えるが，これには以下に述べる条件が必要である．

3 Hilbert変換と直交検波

● 実数信号(複素信号の仮説)

自然界に存在する現象は，実数としてのみ観測可能である．図6の受信側のアンテナに受かる信号も実数である．しかし，先ほども述べたように，実数のままでは位相と振幅をパラメータとして扱うことができず，複素数の形にしてこれが可能になる．中間に実数である電波を介しながらも，複素数の送受信が可能であることも示した．

図7は，ある複素数の信号の軌跡とその実軸への投影を描いたものである．このような複素信号を$x_+(t)$としよう．これは極座標の形式でも表現できて，

$$x_+(t) = a(t) + jb(t) = A(t) \cdot \exp[j(\omega_c t + \theta(t))] \quad \cdots\cdots (23)$$

となる．$A(t)$は信号の振幅であり，$\theta(t)$は位相である．ω_cは前節の搬送波周波数で，信号はこの周波数近辺に展開していることを意味する．この信号の共役複素数(虚数部の符号が逆)があるとして，これを$x_-(t)$とすると，

$$x_-(t) = a(t) - jb(t) = A(t) \cdot \exp[-j(\omega_c t + \theta(t))] \quad \cdots\cdots (24)$$

となる．数式の上では式(24)の信号は，式(23)に対して，マイナスの周波数を持つことになる．式(23)と式(24)を加算すると，実数部だけが残り，$2a(t)$となる．$x_+(t)$と$x_-(t)$を周波数軸に描くと，図8のようになる．$x_+(t)$と$x_-(t)$はどちらも複素数の信号であるが，これが同時に存在すると虚数部が打ち消しあって実数部だけが残る．図9にそのようすを描いてみた．

"純然たる実数の信号というのは，ある意味存在しない．そ

れは，このように鏡に映ったような「負の周波数」の信号との合成として存在している”と考えてみてはどうだろうか．実際問題，実数信号を離散フーリエ変換すると，図8のように，正と負の周波数に対称にスペクトラムが現れることがよく知られている[6]．

本節の冒頭で「自然界の信号は実数のみ」と言っておきながら，「複素数の信号しかない」などと，何やら神学的な展開になってしまったが，次節でその謎解きをしたい．

● Hilbert 変換[注1]

図8の二つの信号のうち，$x_-(t)$だけ消去できれば，残る$x_+(t)$は式(23)のような「立派な」複素信号になるはずである．それには，図10のような，負の周波数成分だけ除去するフィルタがあればよい．

図10の特性$U(\omega)$の“インパルス応答”は，その“逆フーリエ変換”μ_0で求められる．

$$\mu_0(t)=\int_{-\infty}^{\infty}U(\omega)e^{j2\pi\omega t}d\omega=\delta(t)+\frac{j}{\pi t} \quad \cdots\cdots\cdots\cdots (25)$$

信号“$x_+(t)+x_-(t)$”に，式(25)を畳み込み処理で施せば，図10のフィルタをかけたことになる．

$$x_+(t)=(x_+(t)+x_-(t))*\mu_0(t) \text{ ，＊は畳み込みの記号} \cdots (26)$$

となるはずである．式(26)を変形すれば，

$$\left.\begin{aligned} a(t)+j(b)&=a(t)*\left(\delta(t)+\frac{j}{\pi t}\right)\\ &=a(t)+j\left(a(t)*\frac{j}{\pi t}\right)\end{aligned}\right\} \cdots\cdots\cdots\cdots (27)$$

となる．これより，

$$b(t)=a(t)*\frac{j}{\pi t}=\frac{1}{\pi}\int_{-\infty}^{\infty}\frac{a(t)}{t-\tau}d\tau \quad \cdots\cdots\cdots\cdots (28)$$

となる．これは$a(t)$にインパルス応答$h(t)$が，

$$h(t)=\frac{1}{\pi t} \quad \cdots\cdots\cdots\cdots (29)$$

であるフィルタを処理したことに相当する．そのインパルス応答は図11の形となる．

また，式(28)は，Hilbert 変換と呼ばれ，$b(t)$は$a(t)$のHilbert 変換となる．つまり，ある実数信号$a(t)$があり，そのHilbert 変換である式(28)を虚数部とすれば，複素信号式(23)が得られるというわけである．この形であれば，振幅と位相を時系列のパラメータとして扱うことができる〔例：式(10)と式(11)〕．

● 直交検波

しかしながら，図11，式(29)のようなインパルス応答を持つフィルタを厳密な形で実現するのは極めて困難だ．それは，t=0のところで値が無限大になる特異点があるからである．実際に通信の技術で使われるのは，図12のように，

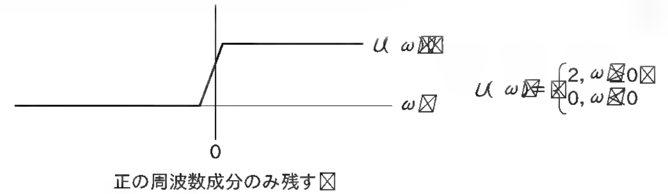


図10　負の周波数成分を除去するフィルタ

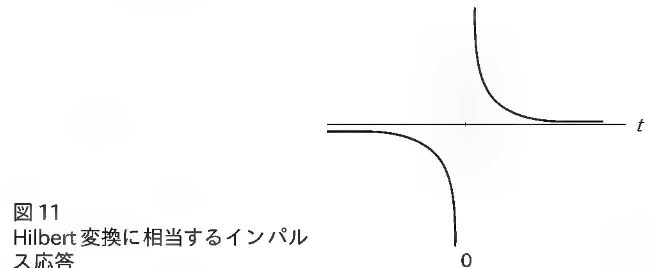


図11
Hilbert 変換に相当するインパルス応答

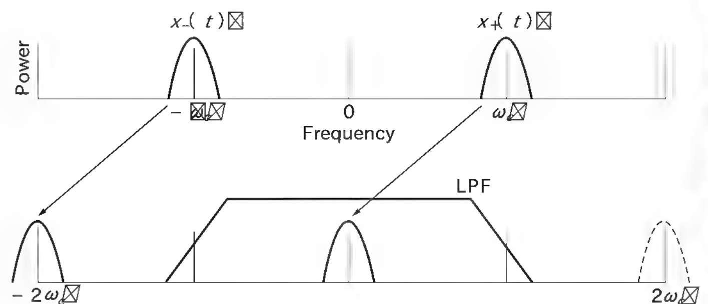


図12　信号の周波数移動と Hilbert 変換対生成

- 信号$x_+(t)$を0Hzに「移動」させる
- $x_-(t)$は0Hzからなるべく遠いところへ移動させる
- ローパス・フィルタで，0Hzに移動した$x_+(t)$だけを残し，これをベースバンド信号とする

という手法を取っている[1]．実は，式(22)を導いた手法はこれである．これが数学的にも正しいことを説明しよう．

図11の$x_+(t)$と$x_-(t)$はどちらも複素数の信号で，式(23)，式(24)である．その搬送波周波数をω_cとする．これを周波数を移動させるためには，単純に$\exp(-j\omega_c)$をかけてやればよい．その結果を$S_0(t)$とすると，

$$S_0(t)=\frac{1}{2}(x_-(t)+x_-(t))\exp[-j\omega_c t] \quad \cdots\cdots\cdots\cdots (30)$$

となる．$x_+(t)$と$x_-(t)$を式(23)，式(24)で置き換えて整理すると，

$$S_0(t)=\frac{1}{2}A(t)\times\left(\exp[j\theta(t)]+\exp[-j(2\omega_c t+\theta(t))]\right) \cdots (31)$$

となる．かっこ内第1項は$x_+(t)$が0Hzに移動したもの．同じく第2項は$x_-(t)$が移動したものだが，これを除去してやればよい．それは「普通」のローパス・フィルタで可能である．式

注1：本節の「インパルス応答」，「フーリエ変換」，「畳み込み」，「デルタ関数$\delta(t)$」などの用語を未学習な読者は，まず文献(6)を学習することをお勧めする．

式(31)はその結果，

$$S_1(t) = \frac{1}{2}A(t)\exp\left[j\theta(t)\right] \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (32)$$

となる．これは式(22)に対応して，

$$S_1(t) = 1/2\{I(t) + jQ(t)\} \times \exp\left[j\theta_0\right] \cdots\cdots\cdots\cdots\cdots (33)$$

となる．これはベースバンド信号がθ_0だけ位相回転したものである．つまり，図6のシステムの受信側は，図12を実行するものということである．

式(33)の$I(t)$と$Q(t)$は，Hilbert変換対の関係にある．このような実数部と虚数部がHilbert変換対の関係にある信号は「解析信号」と呼ばれている[7]．

先に搬送波で伝送される信号を式(12)で表した．式(33)は，これからω_cを取り去り，ベースバンド信号だけを残したことになる．情報を伝達する内容はすべてベースバンド信号(図4はその例)に含まれており，搬送波周波数ω_cは「乗り物」にしかすぎない．図6は送信側でベースバンド信号を乗り物に「乗せ」，受信側で「降ろす」という手続きを行っているわけである．乗せることを「直交変調」といい，降ろすことを「直交検波」と呼ぶ[1]．

竹とんぼの静止している状態の羽根がベースバンド信号であり，これを回転させると飛翔する様を想像すれば理解の助けになると思う．竹とんぼの回転数は，搬送波周波数に対応する．伝達する情報は竹とんぼの羽根の形状にあり，それは搬送波周波数とは無関係である．

● 直交検波の条件

図6および前の節で説明したシステムで，式(32)と式(33)の受信信号を得るためには，$x_+(t)$と$x_-(t)$をきっちり分離しなければならない．ところが図13のように，ω_cが低いと，$x_+(t)$と$x_-(t)$の周波数の「差」が小さく，分離が困難になってしまう．

また，直交検波の処理をディジタルで行う場合は，サンプリング定理により，サンプリング周波数の1/2以下の周波数しか扱えない．図13，図14のω_sはサンプリング周波数(f_s)の角周波数としたものである．図14は，ω_cがω_sより「少しだけ低い」場合である．直交検波による周波数移動の結果，$x_-(t)$の周波数は$-2\omega_c$となるが，これは回り込んで0Hzより「少しだけ高い」ということになり，$x_-(t)$が正の周波数側になっただけで，図13と同じことになる．

このように，$x_+(t)$と$x_-(t)$の分離という観点から見ると，ω_cはω_sの1/4がもっとも有利ということになる．図12はそのように描いたものである．

4 IF信号処理

今までは話が煩雑になることを避けるために触れなかったが，実際の無線機では信号処理部とアンテナの間に，ω_cをより高い搬送波周波数に変換するための，RFアップ・コンバータが送信側に置かれ，受信側では，RFダウン・コンバータが置かれる．今のところ，この部分はアナログの高周波回路で構

COLUMN-01

○○変換

音声や機械振動など比較的低い周波数の信号は図13のように0Hz(DC：直流成分)に近い成分もふんだんに存在する場合が少なくない．はなはだしいときには，0Hzにピークがあったりする．そのような信号の直交検波は，厳密には不可能といってよい．このような場合，多少の誤差を見込んで「正攻法」で図11，式(29)のようなHilbert変換を直接行うこともある[7]．また，FFT(離散高速フーリエ変換)により，スペクトラムを求め，負の周波数成分を0に消去して逆フーリエ変換を行うという手法もある．

概して「○○変換」という処理は，厳密には実現不可能である．フーリエ変換にしても，厳密にいえば無限の過去から無限の未来までの信号を取り込んで，積分しなくてはならない．そのようなことは不可能なので，窓関数という取り扱い可能な一定時間だけ切り出して，フーリエ変換“のようなもの”を計算しているにすぎない．この場合，限定されるのは周波数分解能である．不確定性原理というやつである．何ごとも限定条件の中で誤差を見込みながら処理する．これがディジタル信号処理の一面なのである．

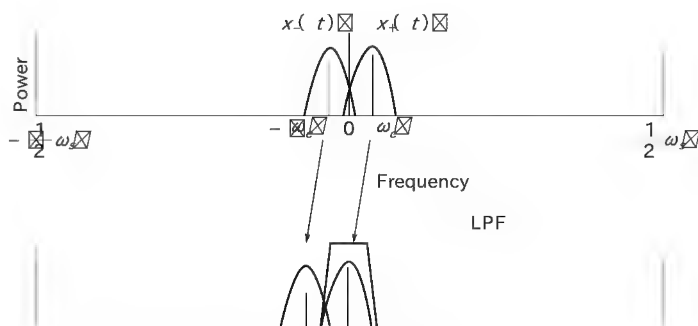


図13　低域信号の周波数と$x_+(t)$と$x_-(t)$の分離

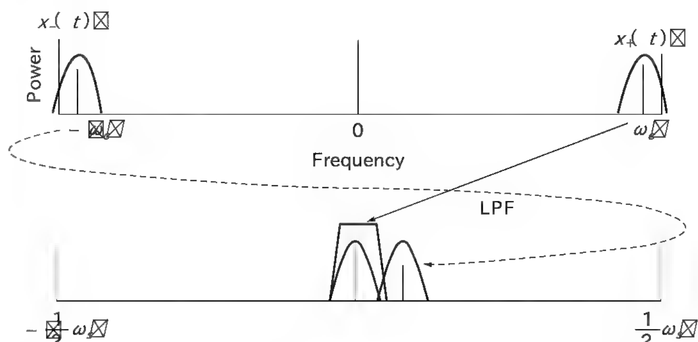


図14　ナイキスト周波数に近い信号の周波数移動と$x_+(t)$と$x_-(t)$の分離

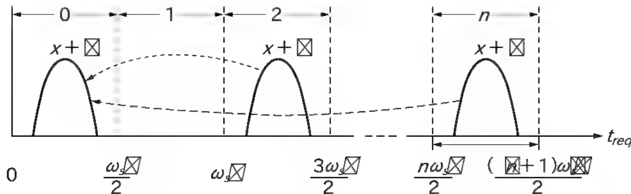


図15　アンダ・サンプリング法

成されている(図6).

②の周波数 ω_c は「IF(中間)周波数」と呼ばれている．この部分の信号はIF信号と呼ばれたり，単に「IF」と呼ばれたりする．また，この部分の信号処理を「IF信号処理」や「IF処理」などと呼ぶ．これは信号処理技術者の仕事である．

RFアップ/ダウン・コンバータは，信号処理技術者の立ち場からいくと，搬送波周波数が ω_c から ω_{RF} に，あるいはその逆に移動するだけで，ブラック・ボックスと考えてさしつかえない．こちらは「高周波技術者」の仕事となる．

空中を伝播する搬送波周波数 ω_{RF} は，通信の用途により周波数が法律などで決められていて，勝手な値を使うことができないが，中間周波数 ω_c のほうは，システムの条件によりつごうのよい周波数を機器の設計者が決めることができる．

● ダウン・コンバータのIFサンプリング法

本章では，ディジタル信号処理の話をしているので，IF信号処理をディジタル信号処理で行うことを前提にする．図6の受信部分でも①～③のどこかでA-D変換器により，信号のディジタル化を行う．当然のことながら，A-D変換器よりアンテナ側はアナログ信号処理になり，その逆側はディジタル信号処理となる．携帯電話器など，小型の機器は①で，基地局や計測器など大型で高性能を要求される場合は③に置かれることが多いようだ．ここでは③の位置でA-D変換する方式について説明する．この方式は「IFサンプリング法」と呼ばれている．

ディジタル信号処理では，サンプリング定理がその大前提になり，周波数はサンプリング周波数 ω_s の1/2以下でしか扱えないことになっている．もしA-D変換器に $\omega_s/2$ 以上の信号が入力さても，$\omega_s/2$ 以下の周波数として観測される．図15で0，1，2，…n，…と示した領域のうち，n 番目の周波数範囲は，

$$\frac{n}{2}\omega_s \le \omega \le \frac{n+1}{2}\omega_s \quad \cdots\cdots (34)$$

であるが，n がいくつであってもすべて，$0 \le \omega \le \omega_s/2$ として観測される．逆にいえば，サンプリングされた信号は，k がいくつであるか判別できないということでもある．

無線通信の世界では，むしろこの現象を利用して，ナイキスト周波数以上の高い周波数の信号をサンプリングする手法が利用されており「アンダ・サンプリング法」と呼ばれている[8]．ただし，この方式の場合，信号は特定の n に対応した周波数領域に帯域制限されていなければならない．また，A-D変換器の入力の応答周波数にも限界があるので，n をむやみに大きくすることもできない．n は0から3程度が一般的な使われ方だろう．

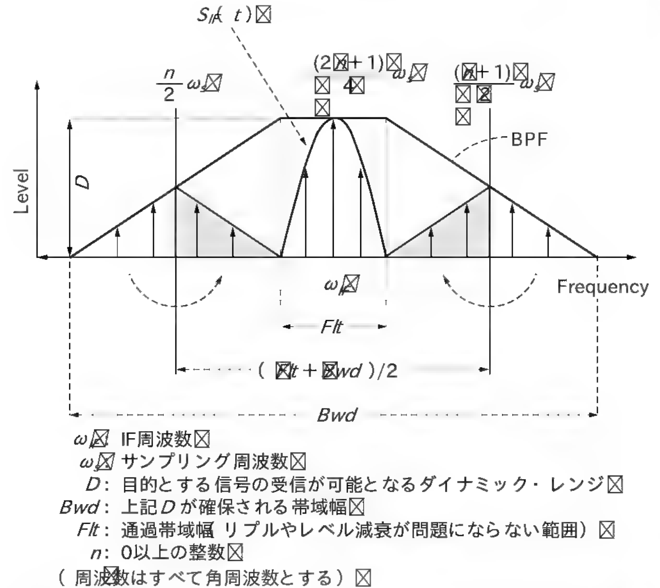


図16　IFバンド・パス・フィルタの周波数特性と得られるスペクトル

IFサンプリングが正常に機能するためには，A-D変換器の前で信号が特定の n の部分に制限されていなければならない．このために，バンド・パス・フィルタをRFダウン・コンバータの出力に置く．これを「IFバンド・パス・フィルタ」とする．図16はその特性の概念を表したものである．

理想的には，式(34)の範囲だけ通すフィルタであればよいのだが，現実には不可能である．過渡帯域はいくらかの傾斜がつくことになる．通過特性が式(34)の範囲からはみ出した部分は，$\frac{n}{2}\omega_s$，$\frac{n+1}{2}\omega_s$ を中心に鏡のように降り返し(エイリアシング)が発生する．図16の三角形の灰色の部分である．この部分は降り返しによる不用なノイズが混入している可能性がある．残った部分の帯域がフィルタの通過帯域である Flt より狭くならないのが理想である．

図中で D は信号処理に必要とされるダイナミック・レンジ．Bwd は D に対応する帯域幅である．Flt と Bwd はフィルタの特性としてセットで決定される．これらとサンプリング周波数との関係は，

$$\omega_s \ge (Flt + Bwd) \quad \cdots\cdots (35)$$

となる．ただしこの関係は，バンド・パス・フィルタの中心周波数が，$\frac{n}{2}\omega_s$ と $\frac{n+1}{2}\omega_s$ の中間の $\frac{2n+1}{2}\omega_s$ であること．フィルタの特性が左右対称であるという条件が付く．これらの条件が満たされない場合は，サンプリング周波数を高くするか，Flt を狭くするなどの妥協が必要になる．なお，折り返しの部分と通過帯域の関係には十分に注意を払わなければならない．

● アップ・コンバータ

図6の送信側のシステムは，受信側と逆の動作を行う．要す

> **COLUMN-02**
>
> ## 信号の送信と受信
>
> 一般論として，信号を発生することよりも，受信することのほうが条件が厳しくなる．送信器の場合にはシステムを通過する信号はすべて「自分で作り出す」のに対して，受信器は「入力に何が入ってくるかわからない」からだ．つまり，空中を飛び交う電波から所望の信号だけを選別するために，フィルタが必要になる．逆に，受信装置でも．実験室の中などで入力信号が特定の帯域に限定されていることがわかっていれば，IF にバンド・パス・フィルタは不用になる．

るに式(13)の信号，

$$S_{IF}(t)=I(t)\cdot\cos(\omega_c t)-Q(t)\cdot\sin(\omega_c t) \quad \cdots\cdots\cdots\cdots (36)$$

を発生する．注意事項としては，

- D-A 変換器は図6の①，②，③のどこに設置してもよい
- D-A 変換器の出力は盛大な高調波成分を持っているので，これを除去するローパス・フィルタを施す
- 図16と同様に，信号の帯域幅に対応したサンプリング周波数が十分確保されていること

などがある．ディジタル・アップ・コンバータについては，次の項でさらに詳しく述べる．

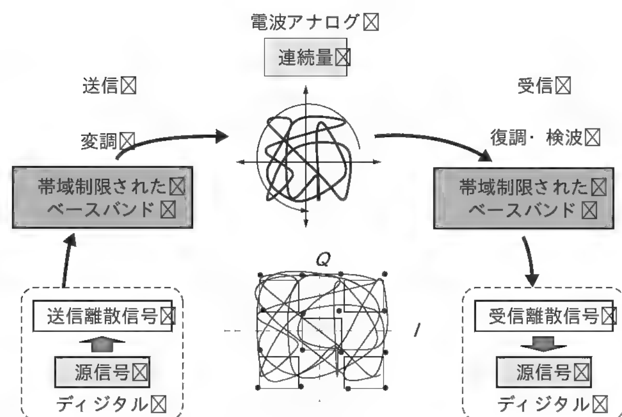


図17　ディジタル無線信号の変遷

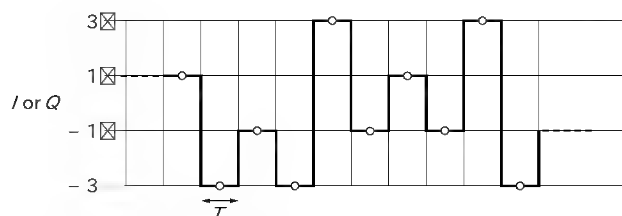


図18　I または Q の送信離散信号

5 ディジタル無線通信の信号の帯域制限

ここでもう一度，図4を眺めていただきたい．図中の●は「シンボル点」と呼ばれており，“シンボル・レート”と呼ばれる一定の時間間隔ごとに，信号はここを通過するようになっている．各シンボル点は方式ごとに規定の位置が定められている．16QAM の場合は，I と Q は±1または±3のいずれかの値を取ることになっており，シンボル点としては16通りの可能性がある．したがって，1シンボルあたり4ビットの情報を持たせることができる．

π/4DQPSK の場合は，シンボル間の位相の変化が±45度または±135度の4通りと決められており，1シンボルあたり2ビットの情報を伝達することができる[1]．

ディジタル無線の場合，このような離散的な I と Q の推移に帯域制限を施し，連続的な式(6)と式(13)の形式に変換したのち，アンテナから信号を送信し，受信はその逆の処理を行うのが一般的であろう．そのメカニズムについては既に述べた．その概念を図17に示す．

● 送信器における帯域制限

図17の「源信号」は，直接送受信するデータだが，そのままでは変調に適合しないので，変調方式に適合させた信号である「送信離散量」に変換される．1シンボルあたり1ビットであったり，4ビットであったりする．図18は16QAM 変調の場合の I または Q の時系列の例である．小さな○はシンボル点であり，1，3，−1，−3の値のどれかをシンボル・レートに対応する時間間隔で変化していく．

しかし，このような矩形パルスの集合のような信号は，次式のような非常に広帯域な高調波成分をもっている．

$$|F(\omega)|=A\frac{\sin(\omega t)}{\omega} \text{，} A\text{は比例定数} \quad \cdots\cdots\cdots\cdots\cdots (37)$$

図19に矩形波の時間軸と周波数軸の形状の例を示した[9]．サイド・ローブは上下2本目までしか描いていないが，実際には無限のかなたまで徐々に下がりながら伸びている．このような信号は，ほかのチャネルとの混信を防ぐため，また，周波数の有効利用，電力の節約などのために，帯域制限を行う必要があ

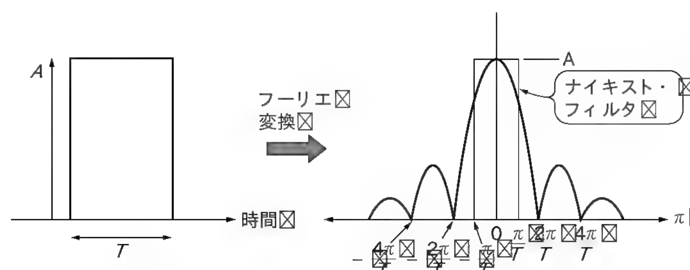


図19　矩形信号の周波数成分

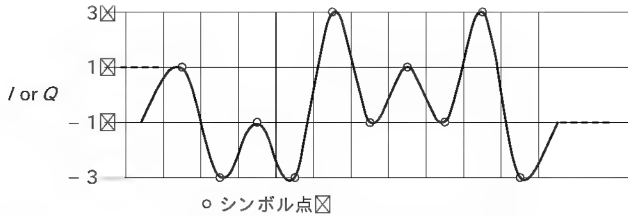


図20　ロール・オフ・フィルタを通過した I, Q信号

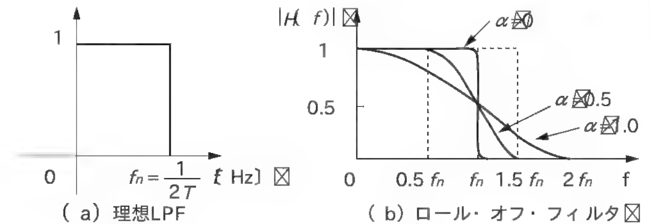


図21　コサイン・ロール・オフ・フィルタ

る．図6のLPFがそれである．このためによく使われるのが図19のロール・オフ・フィルタで示された特性をもつもので，これは

$$-\frac{\pi}{T} \leq \omega \leq \frac{\pi}{T} \quad \cdots\cdots(38)$$

の範囲だけを通過させる「ナイキスト・フィルタ」と呼ばれるフィルタである[9]．この種のフィルタを通過させると，図18の信号は，図20のようになる．シンボル点の位置を動かすことなく信号を滑らかに（よって帯域幅を狭くする）する特性をもっている．

通過帯域としては，図19のようにメイン・ローブ（中央のもっとも大きなピーク）の半分の周波数帯域で十分なことにも留意してほしい．このようなフィルタの設計も信号処理技術者の重要な仕事である．もちろん，図のような矩形の特性のフィルタを作ることは不可能なので，よく使われるのが「コサイン・ロール・オフ・フィルタ」という，図21（b）のような特性のフィルタである．係数 $\alpha=0$ で理想フィルタで，$\alpha=1$ の場合は0Hzから減衰が始まる特性になる．このフィルタの周波数特性は次の式の $H(\omega)$ で表される．

$$H(\omega)=\begin{cases} T, & \left(0 \leq |f| \leq \dfrac{1-\alpha}{2T}\right) \\ \dfrac{T}{2}\left\{1+\cos\left[\dfrac{\pi T}{\alpha}\left(|f|-\dfrac{1-\alpha}{2T}\right)\right]\right\}, & \left(\dfrac{1-\alpha}{2T} \leq |f| \leq \dfrac{1+\alpha}{2T}\right) \\ 0, & \left(\dfrac{1+\alpha}{2T} \leq |f|\right) \end{cases} \quad \cdots\cdots(39)$$

中段の過渡帯域の通過特性は，コサイン関数に1を足した形になっている．図17に描いたように，このように「滑らか」にされた信号がアンテナから電波となって放射される．

● 受信器の帯域制限

図17に描いたように，受信側でも帯域制限が必要である．それは先にも述べたように，直交検波のため $x_-(t)$ 信号の除去がまず第一である．次に，空中には隣接のチャネルなど，さまざまな信号が飛びかっている，これから，受信の対象である単一チャネルだけを選別するために帯域制限フィルタが必要である．このフィルタを通った信号は，図20と同じものになるはずである（多少のノイズが加わるが）．ここからシンボル点の I, Q 値を取り出せば注2，送信離散信号に対応する「離散受信信号」が得られる．これと送信と逆のフォーマット変換を行えば，源信号が得られる．このように，ディジタル無線では，離散量－帯域制限－アナログ連続量という信号の変遷を行わせている．

● ルート・ナイキスト・フィルタによる送信と受信

送信時にナイキスト・フィルタが必要であるが，受信機にも帯域制限フィルタが必要である．ならば送信と受信でナイキスト・フィルタを等当分に振り分けよう，というのが「ルート・ナイキスト・フィルタ」と呼ばれるものである．周波数特性は式（39）の平方根となる．

$$\left\{1+\cos\left[\frac{\pi T}{\alpha}\left(|f|-\frac{1-\alpha}{2T}\right)\right]\right\}=2\cos\left[\frac{\pi T}{2\alpha}\left(|f|-\frac{1-\alpha}{2T}\right)\right]^2 \quad \cdots\cdots(40)$$

であるので，

$$M(\omega)=\sqrt{H(\omega)}=\begin{cases} T \\ \sqrt{2}\cos\left[\dfrac{\pi T}{2\alpha}\left(|f|-\dfrac{1-\alpha}{2T}\right)\right] \\ 0 \end{cases} \quad \cdots\cdots(41)$$

となる．なお，三つの周波数範囲は，式（39）と同じである．

このような特性のフィルタを，送信と受信でそれぞれ施せば，トータルの特性が式（38）となって受信側でナイキスト・フィルタが成立し，シンボル点の受信が可能となる．

● 信号の帯域幅とサンプリング・レート

$S(t)=I(t)+jQ(t)$ の形式で表現できる信号のサンプリング・レート（周波数）が f_s であるとき，それが扱うことのできる帯域幅 BW は，ナイキストの定理により，最大で f_s である．負の周波数も表現できるからである．ただし，ナイキスト周波数（$\pm f_s/2$）近辺では，無限長の時間があってはじめて信号表現可能となり，それは実用的ではない．現実的には $BW <$ 0.8× f_s 程度が目安ではなかろうか．

6　送受信機の実現方法，アナログとディジタルの境界

図22（a）は，ディジタル化された送受信機のブロック図で

注2：そのためには，シンボルと同期を取ったタイミングで I, Q 値をサンプリングする必要がある．同期については後述する．

ある．送受信兼用で描いた信号は左右両方の方向に流れる．左から右が受信機で，右から左に流れると送信機の処理になる．前項で述べた帯域制限は，この図ではアナログのLPFで行っている．この方式だと，A-DおよびD-A変換器は比較的低速のもので対応でき，システム全体が小型で安価にできる．しかし，いったん作ってしまうと仕様の変更は不可能である．携帯電話など小型の無線機は，この形を取ることが多いようだ．最近はこの図の大部分の要素が1チップに収めたものもあるが，その場合，用途は特化されたものになる．

図22(c)のA-Dは，アナログIF信号を入力として，IFサンプリング法を用い，直交検波変調以下のベースバンド処理をディジタル信号処理で行う．同様に図22(b)はディジタル化されたディジタル送信機のブロック図である．大きな四角で囲まれた部分は，それぞれ，DDC(Digital Down Converter)とDUC(Digital Up Converter)と呼ばれている．図23はDDC，DUCを組み込んだ無線システムの例である．DDCおよびDDCは専用のLSIとして市販されている．また，FPGAのIPソースとしても市販されている．今やこの種のデバイスを使いこなすことが無線システム設計の重要な要素となっている．

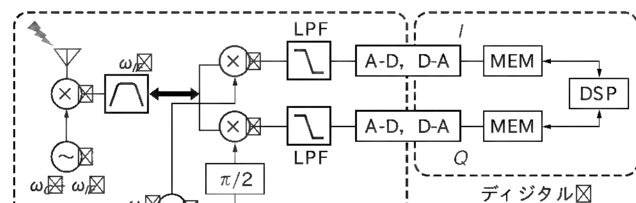
● ディジタル・ダウン・コンバータ(DDC)

図24は，DDCの構成の一例である．アナログ回路による直交検波の処理を数値計算で置き換えるということを行う．アナログ直交検波とDDCの比較を表1に示す．

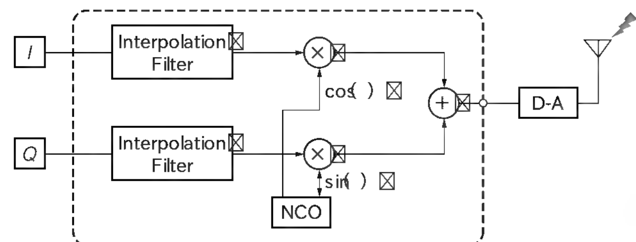
DDCのデメリットというと部品コストにつきるが，これも年々安くなる傾向にある．DDCの各部の動作を入力側からざっと説明する．

▶ 入力

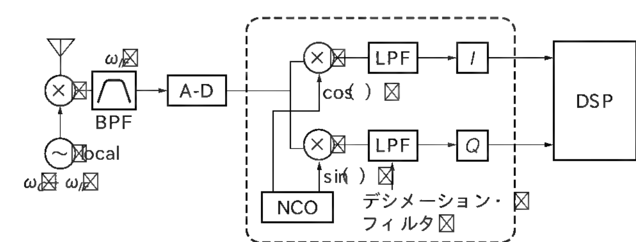
DDCの入力は通常はA-D変換器(ADC)から与えられる．最近ではADCを内蔵した製品もある．ADCにDDCが付いた



(a) ディジタル化された送受信器(アナログが主)



(b) ディジタル・アップ・コンバータを使った送信機



(c) ディジタル・ダウン・コンバータを使った受信機

図22　送受信器の構成

表1　直交検波器のアナログとDDCの比較

	アナログ	DDC
Local周波数精度	部品精度による	クロック源の精度に依存
π/2位相差精度	調整による	完全
sin( )，cos( )レベル精度	調整による	完全
特性の温度・経時変化	ある	ない
部品コスト	安い	高い？
製造時の調整	必要	不要

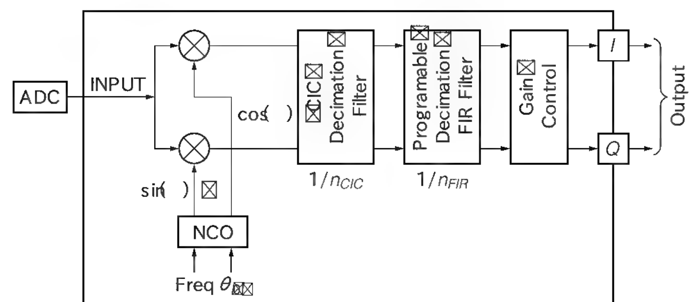


図24　DDCの構成要素

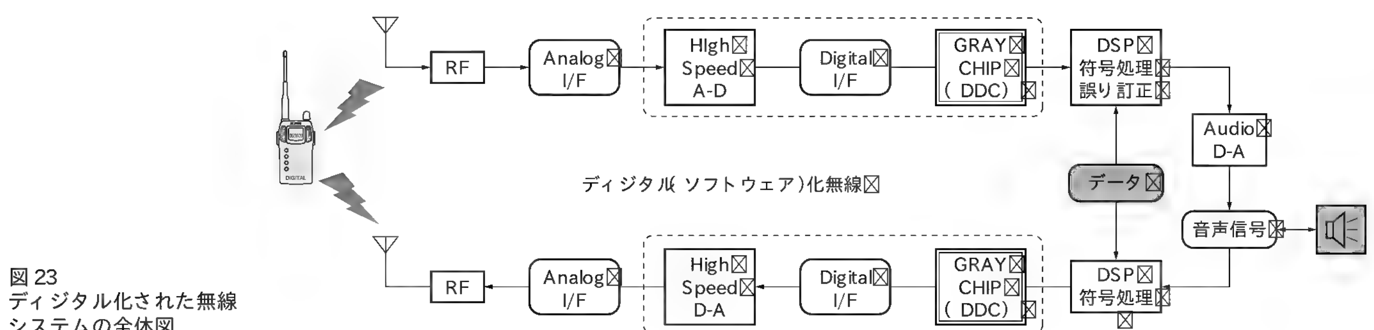


図23
ディジタル化された無線システムの全体図

とも，DDCにADCが付いたとも，どちらともとれる．入力の最高サンプリング周波数は100MHz程度が主流である．新しいものでは150MHzのものもある[10]．このくらいのレートの場合，従来の無線機のアナログIF信号の処理にも余裕をもって対応できる．

アナログ回路を簡素にするためには，ADCのサンプリング周波数はなるべく高いほうが有利である．また，高いダイナミック・レンジで信号を扱うには，高いサンプリング・レートでA-D変換を行い，後述のデシメーション・フィルタ処理を施すと平均化効果により高いダイナミック・レンジを得ることができる．

▶ NCO（Numerical Controlled Oscillator）

数値演算的にsin（），cos（）関数を発生する．アナログの局部発振器に相当する．通常は入力として，FREQ（周波数）とPHASE（初期位相）を持つ．発生周波数はナイキスト周波数（クロック周波数の1/2）以下で可能．設定分解能はFREQの入力ビット数に依存するが，市販品では32ビットや48ビットなど，十二分な分解能をもたせている．

ここでのsin（），cos（）の位相差の精度や，レベルの一致度，その後の処理の精度を決める重要なパラメータになる．アナログでは測定器を使いながら調整しなければならないが，ディジタルでは，この点は完璧なものが得られる．

▶ ミキサ

図24の⊗で示したところで，DDCが直交検波のために NCOで発生したsin（），cos（）と入力の乗算を行う．それぞれの出力をIとQ成分とすると，この出力の段階での信号は $x_+(t)$ と $x_-(t)$ が混在しており，図25のようになっている．$x_+(t)$ の周波数の低くてゆるやかな動きと，$x_-(t)$ の周波数の高い成分によって1ポイントごとに上下に激しく動く成分が混在している．この信号は，次のCICフィルタとFIRフィルタでデシメーション処理される．

▶ ローパス・フィルタとデシメーション

ミキサの出力はA-Dのナイキスト周波数に相当する広い周波数帯域を持っている．ここから $x_+(t)$ の信号を取り出すが，サンプリング周波数は信号の帯域幅と同程度で十分である．そこで，ローパス・フィルタの周波数を一つおき，二つおき，あるいはもっと少なく「間引き」して出力サンプリング数を減らす．この処理を「デシメーション」と呼ぶ．ディジタル無線では，シンボル点のみのサンプル・レートまでデシメーションすることが一つの目標となる．たとえば，図20のシンボル点だけ残せばよいのである．

また，ダイナミック・レンジを得るためには，なるべく高い周波数でサンプリングを行ったのちに，狭帯域のローパス・フィルタで信号の帯域を狭め，高い率でデシメーションを行う．平均化効果によりA-D出力データより高いダイナミック・レンジが得られる．

▶ CICフィルタ（Cascade Integrator Comb Filters）

場合によっては，この部分ではなく，次のFIRフィルタに直接信号が入力される場合もある[10]．

図26はCICデシメーション・フィルタの構成の一例であ

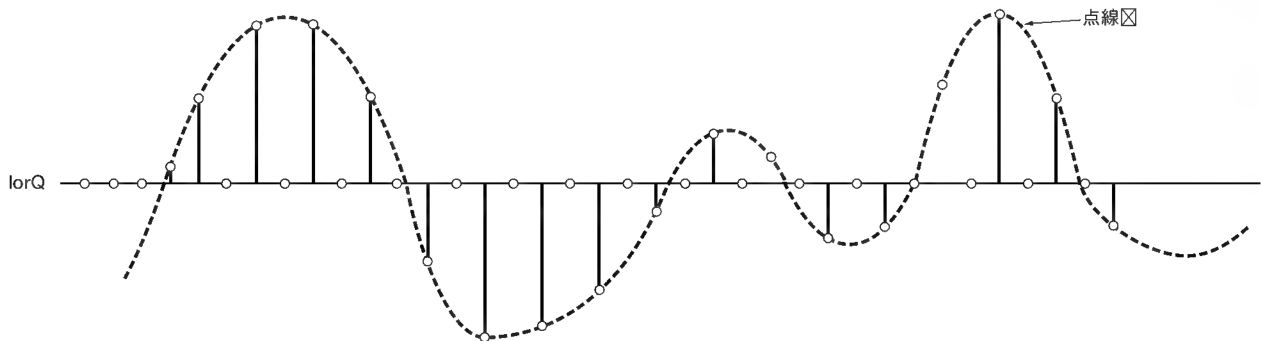


図25　ミキサ出力信号の状態

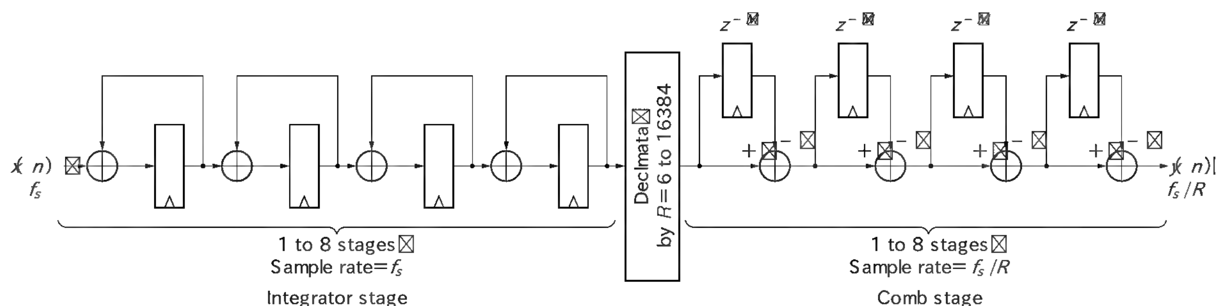


図26　CICデシメーション・フィルタの構成

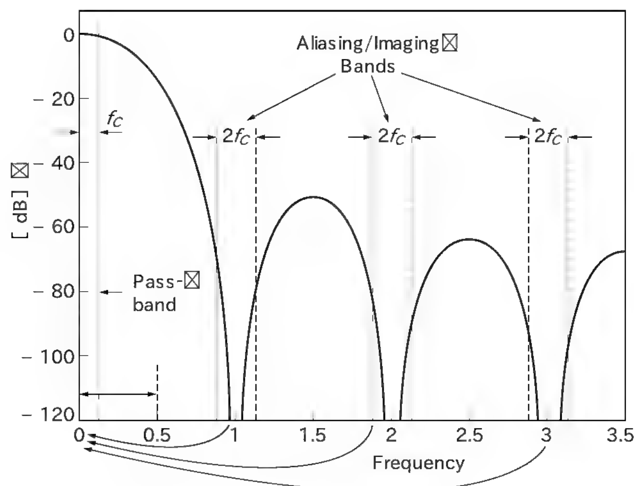


図27　CICフィルタの周波数特性

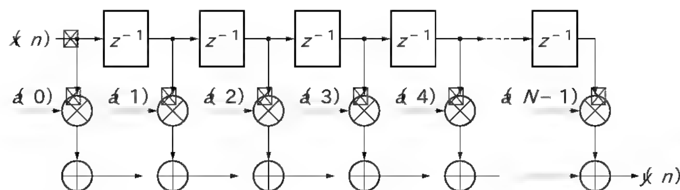


図28　FIRフィルタ

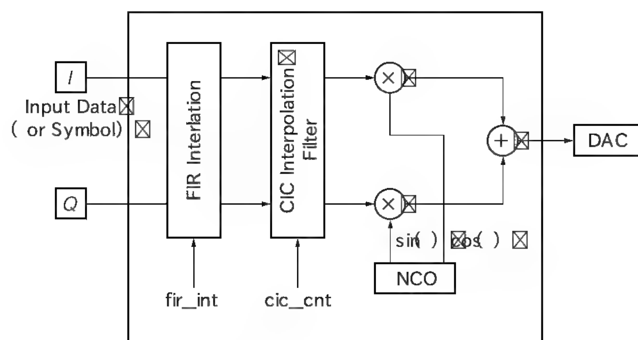


図29　DUCの構成要素

る．参考文献(2)に簡単な解説が，参考文献(11)に詳しい報告がある．

CICフィルタは「くし型フィルタ」ともいわれ，図27のようなピークが一定間隔で並ぶという特性をもっている．詳しい説明は割愛するが，大ざっぱにいうと“足して割る”という動作の組み合わせでローパス・フィルタとデシメーションを行う．特性は，

$$|H(f)| = \left[\frac{\sin \pi M f}{\sin \frac{\pi f}{R}} \right]^{2N} \quad \cdots\cdots (42)$$

と近似することができる．ここでRはデシメーション率，Mは任意の整数，NはCICの段数(1～8程度)を示す．また，周波数fは，“f=サンプリング周波数/R”で正規化したものとする．

式(42)は，sinc関数(sin(x)/x)のべき乗の形であり，f=1/Mの整数倍の周波数にNULL点を有しており，図27はその例である．デシメーションの結果，降り返しが発生してもちょうどこれらのNULL点が0Hzに回り込むようになる．この現象を利用して，簡単な演算，つまり簡単な回路で高い率のデシメーションが可能になる．しかも回路構成が簡単なので，高速動作もできる…と，まことに巧妙なカラクリである．

DDCのデシメーションは，最初のCICでサンプリング・レートを落とし，次のFIRフィルタで精密な特性を得るという手順が順当である．

▶ FIRフィルタ(Finite Impulse Response Filter)

図28はFIRフィルタの概念を描いたものである．xは入力，yは出力で，N個の係数があり，

$$y(k) = \sum_{n=0}^{N-1} a(n) x(k-n), \quad k = 0, 1, \cdots \quad \cdots\cdots (43)$$

以上のように積和演算を行う．Nをフィルタのタップ数と呼ぶ．基本的に入力と出力のサンプリング・レートは同じである．その場合，1サンプル周期に乗算と加算(積和)をN回実行しなくてはならない．デシメーションを行う場合は，yを間引いて出力すればよい．前段のCICフィルタでデシメーションが十分なされていれば，1個の乗算器と加算器でも時分割処理により，積和演算を実行できる場合もある．

リアルタイムに積和を実行しようとして，図28の構成をそのまま実装すると，TAPの数だけ乗算器と加算器が必要になり，回路規模が非常に大きくなってしまう．そこで，“Distributed Arithmetic”(以下DA)という方式が提案されている[12][13][14]．詳細は割愛するが，乗算を実際に行うのではなく，積和結果をテーブルから参照する方式である[2]．

CICフィルタもそうであるが，この種のディジタル・フィルタは乗算を直接行うことなく，特殊な論理回路で構成されていて，DSPとはまったく違う構造になっている．同じ処理をDSPで行おうとすると，数十個，あるいは数百個必要になるだろう．

● ディジタル・アップ・コンバータ(DUC)

図29は，DUCの構成例である．DUCは，DDCと逆の処理で，シンボル・データなどの低速のベースバンド信号を，IF信号に相当する信号まで補間処理してサンプリング・レートを高める．

図29の例では，入力はIとQのベースバンド信号である，ときにはシンボル・データそのものという場合もある．構成要素のFIRフィルタ，CICフィルタ，ミキサ，NCOはDDCと同じものだが，二つのフィルタは補間処理に使われる．補間処理の概略を図30に表す．入力は，サンプル・レートの低い信号である．ディジタル通信機の場合，シンボル・データそのものの場合が多いだろう．この各入力の間にゼロのデータを挿入する．補間倍率は，

補間倍率＝入力間に挿入するゼロデータの個数＋1 ……………………………(44)

となる．図では3個挿入しているので，4倍補間になる．この信号にローパス・フィルタを施して出力を得る．

フィルタの特性にナイキスト・フィルタを採用すると，入力データの値に変動がなく中間の値は入力の補間となる．FIRフィルタでは，これは容易にきっちり実現できる．

CICフィルタだが，図27のように「くし型」の名のとおりに随分デコボコの特性である．このようなフィルタで補間ができるのか？と疑問に思われる方もいるかもしれないが，心配はいらない．図29では，CICの入力はその前のFIRフィルタで周波数の低い部分にしか信号がないうえに，ゼロを挿入することによって発生する周波数成分は，ちょうどCICのNULL点近辺の周波数になるからだ．その結果，補間がきっちり実現されることになる．ただし，通過特性は図27のように平たんではないので，FIRフィルタでこれを補正する必要がある．

● ゲイン・コントロール

勘のよい人は，図30を見て，ゼロ挿入された信号にフィルタをかければ，信号のレベルが落ちてしまうことに気が付くかと思う．ゼロ挿入＋LPFによる補間処理は，ゼロ挿入に対応した分，ゲインを上げないと信号のレベルが下がってしまう（ゼロで信号が薄まる）ので，これを補正しなくてはならない，これがゲイン調整である．図30の場合は4倍補間なので，出力レベルを4倍にすれば元のレベルが維持される．

DDCでも同様にゲインの変動に注意し，必要があればこれを補正しなくてはならない．

7 ソフトウェア無線 ——信号処理デバイスの分担領域

図31をご覧いただきたい．近年はA-D，D-A変換器の多ビット高速化が進んだことと（2004年前半で14ビット120MHzのものがある），DDC，DUCも高性能で安価になってきた．これらのデバイスを活用すれば，アナログI/Fを30MHz，あるいは90MHz，帯域幅を最大で30MHz程度に設定し，それ以下の処理はすべてディジタル信号処理に置き換えることが可能になった．DDC，DUCはA-D，D-A変換器と一体のもので，これらはディジタルとアナログの「橋渡し」を行っていると捉えてみてはどうだろうか？　高速な信号処理はDDC，DUCなどのアナログに近いの専用デバイスで処理し，ややレートの下がった部分で複雑な処理をDSPで行うという方法が一般的な流れのようだ．また，中間の機械的処理はDSPの前にFPGAで論理回路を組んで処理してしまうことも多いだろう．図31の左側は高速で単純で，右側ほど低速だが複雑な処理となり，場所に応じて適切なデバイスを採用することが必要になる．

DSPの処理速度が十分に速ければ本章で表した内容がすべてソフトウェアで実現できるわけだが，現在のところそれはまだ無理なようだ．kHzのオーダの信号はDSPで，MHzの信号は専用デバイスで，というのが大ざっぱな使い分けだと思う．

8 同期

ディジタル無線の復調の場合，図20のようにシンボル点は

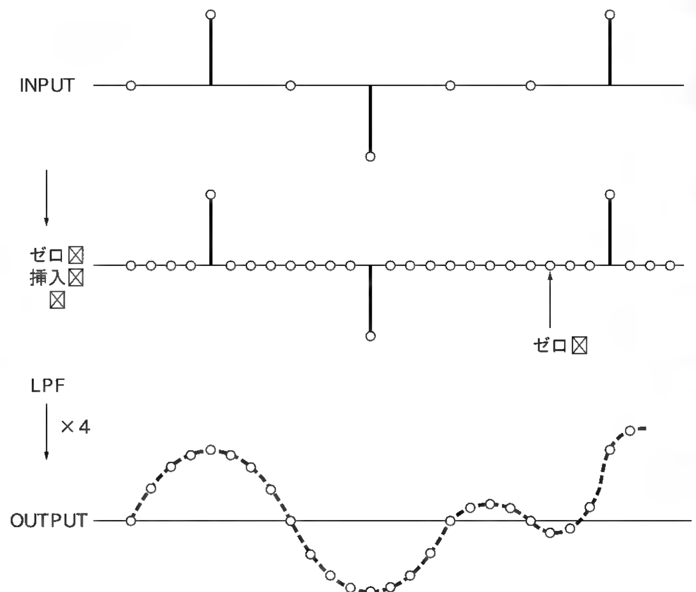


図30　補間フィルタ

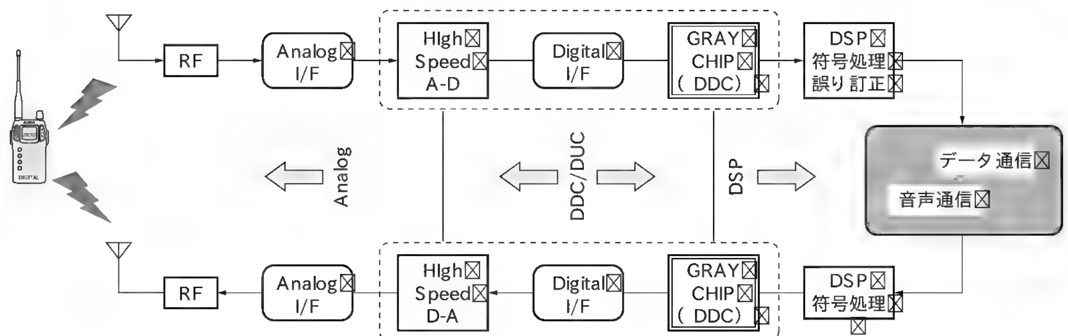


図31
ディジタル化された無線機

連続するアナログ信号の中に"隠されて"いるので，これと同期のかかったサンプリングを行わないとデータの復調はできない．信号の帯域幅に対して十分なサンプリング周波数を確保して(シンボル・レートの2倍が最低限であろう[1])信号を扱っていれば，ソフトウェア的にシンボル点を抽出することは原理的には可能であるが，第2世代の携帯電話ならともかく，最近のワイド・バンド通信になるとDSPの処理速度ではとても間に合わない．現実には，ハードウェアに比重を置いた実現方法をとっているケースが多いようだ[15][16]．

ともあれ，同期のためにやらなくてはならないことは．以下のとおりである．

▶ キャリア推定

搬送波周波数を特定する．実際の受信波はフェージングといって伝播経路の状態の変化で搬送波周波数は変動している．これをなるべくリアルタイムで捕捉しなくてはならない．原理的には，復調した信号の周波数を抽出し，適当な時間平均をとり，これが0Hzになるようにすればよい．

▶ クロック再生とシンボル同期

シンボルの周波数を検出する．通信方式によってシンボル周波数は決まっているので.受信側で発信器を持っていればよい．しかし，初期位相が未知なので受信信号に同期を取られなくてはならないし，ドップラ現象などがあれば周波数も多少の変動が起きるので，これを捕捉する必要がある．同期が取れたらA-D変換のクロックに反映させれば，デシメーションの結果がシンボルになるはずである．

同期のために必要なことは，おおまかにこの2点だが，実現方法は変調方式によりさまざまである．

図4(b)のPSK(位相変調)系の変調では，その振幅(I^2+Q^2)がシンボルに同期して変動する性質があるので，これを利用してPLLをかけて同期を取る．

図4(a)のQAM系では，振幅の変動が複雑なので，上記の方法ではうまくいかない．信号の中に一定間隔で「同期シンボル」を挿入し，これを復調側で検出して同期を取る．どちらの方式も受信した信号の「自己相関」を取り，そのピークが得られるタイミングでクロック再生を行う．

DSPでこれらが実現可能であれば，それが理想である，しかし，通信ではリアルタイムな処理が必須となる．この種の処理がDSPで間に合うかというと，なかなか難しいところである．計測器のようにリアルタイム性が必ずしも重要ではない場合，ハードウェアによらずに，コテコテのソフトウェアで組んだという経験が筆者にはあるが，精密だが処理時間が長くかかってしまい，通信機には通用しない方法であった．

おわりに

ディジタル通信と信号処理をテーマにした書物は多数出版されている．しかしながら，前提とする知識のレベルがかなり高く，学生諸君や新社会人には難物であると思う．今回は，それらの書物では省略されてしまっている部分に重点をおき，読者が次のステップに進む助けになるようにと心掛けながら執筆した．特に，Hilbert変換関係については，わかりやすく説明されているものにお目にかかったことがなく，自分の実務経験から得た理解を書き下ろした．

通信用のデバイスの進歩もめざましく，どんどん高速になっている．アナログとディジタルの境界線をどこに置くかという問いも，その解は半年で変わってしまうくらいだ．本章がその思考の助けになれば幸いである．

参考文献

(1) 長野昌生；デジタル無線通信における信号処理の基礎，Interface，2002年9月号，CQ出版．
(2) 長野昌生；ディジタル無線通信における高速演算デバイス，高速信号処理応用技術学会誌(日刊工業新聞社電子技術誌掲載)，2003年3月．
(3) F.R.Conner Conner"Modulation"1982，Edward Arnold(Publishers) Ltd,「変調入門」高原幹夫訳，森北出版(株)1985.
(4) 岩波数学公式集Ⅱ，岩波書店．
(5) Jhon G. Porakis；ディジタル信号処理Ⅰ，Ⅱ，浜田，田口ら訳，科学技術出版社．
(6) Bringham；高速フーリエ変換，科学技術出版．
(7) Oppenheim，Schafer；ディジタル信号処理(下)，伊達玄訳，1978，コロナ社．
(8) 須崎寛則；高速・高精度周波数計測への応用－超音波ドプラー・ソナーへの応用－，高速信号処理応用技術学会誌(日刊工業新聞社電子技術誌掲載)第2巻第3号，1999年9月．
(9) 尾知博；期待が高まるディジタル通信技術の基礎，Interface，2001年10月号，CQ出版．
(10) GC5016 Data Sheet，Texas Instruments 2004年
(11) E.B.Hogenauer，"*An Economical Class of Digital Filters for Decimation and Interpolation*"，IEEE. Trans. Acoust.，Speech Signal Processing，Vol.29，No.1，pp.155-162 April 1981.
(12) Peled and B. Liu，"*A New Hardware Realization of Digital Filters*"，IEEE Trans. on Acoust.，Speech，Signal Processing，vol. ASSP-22，pp. 456-462，Dec. 1974.
(13) S. A. White，"*Applications of Distributed Arithmetic to Digital Signal Processing*"，IEEE ASSP Magazine，Vol. 6(3)，pp. 4-19，July 1989.
(14) Distributed Arithmetic FIR Filter V7.0：Xilinx data sheet
(15) J.G.Porakis；ディジタルコミュニケーション，坂庭ら訳，1999年7月，科学技術出版社．
(16) 西村芳一；無線によるデータ変復調技術，Design Wave Books，2002年9月，CQ出版．
(17) 河野隆二他；ソフトウェア無線の現状，電通学会B論文誌，2001年7月．

ながの・まさお (株)高速信号処理研究所
nagano@dcplab.co.jp
http://dcplab.co.jp/

Appendix プログラマブル・デバイスか プロセッサか，それともASSP？

実装の心得と勘所

ディジタル信号処理技術を実装する際には，大まかに，(1)PLDやFPGAといったプログラマブル・デバイスを使う，(2)CPUやDSPなどのプロセッサを使う，(3)ASSPを使う――といった三つの方法がある．これらの方法はいずれもトレード・オフの関係にあり，どれを選択するかは開発するアプリケーションやコスト，開発現場の状況なども含めて検討しなければならない．このときの心得や勘所について述べていく．（編集部）

大久 信広

● ディジタル信号処理の三つの実現方法

ディジタル信号処理の実現には，いくつもの方法がある．そして，どのような方法を選択するかによって，コストや性能が大幅に異なってくる．具体的には回路をどのようなデバイスで設計するか，開発ツールが利用可能かどうか，ツールの習熟度はどうか，開発メンバの構成はどうか，デバッグの方法，メンテナンスはどうするかなど，開発現場での環境やユーザの使用方法も含めて慎重に選択しなくてはならない．大きく分けるとPLDやCPLD，FPGAのようなプログラマブル・デバイスを使用する方法と，CPUやDSPといったプロセッサを使用する方法，ASSPを使用する方法の三つの方法から選ぶことができる．

非常に大ざっぱで古典的な考え方として，信号処理の中に乗算を含む場合にはプロセッサまたはASSPを，含まない場合にはプログラマブル・デバイスを使うという分け方がある．つまり，複雑な数学演算を多用する処理については乗算回路やそのほかの演算回路が組み込んであるCPUやDSP，ASSPといったプロセッサに任せて，一方，画像の2値化のように高速であるが単純な処理はプログラマブル・デバイスが受け持つようにするとむだがない．

● コストとの兼ね合い

最近のFPGAの機能の向上と価格の低下には目を見張るものがある．それにもかかわらず，一般的にFPGAやCPLDは専用のCPUやDSP，ASSPに比べ同一の機能を実現するにあたって，コストの点でこれを超えることは不可能だと思われる．この意味で上の考え方は現在に至っても正しいと言えるが，次の場合に限って言えば乗算回路やそのほかの演算回路が含まれていてもプログラマブル・デバイスを利用する価値がある．

- プロトタイプを開発する場合
- 同一回路を複数個必要とする場合
- 非常に処理が高速な場合
- 既存のASSPが存在しなくて，かつ回路全体をコンパクトに作らなければならない場合

プロトタイプ作成時にはプログラマブル・デバイスは非常に大きな助けとなる．大規模な回路では，複数個のプログラマブル・デバイスを組み合わせて使用することもある．プロトタイプを開発する際，往々にして仕様が確定していないとか，最適なアーキテクチャが不明であるといったさまざまな不確定要素が存在する場合がある．開発期間の制限がきびしい場合は，とにかく作業を進めなくてはならないが，プログラマブル・デバイスならどのような変更があってもまず対応に困ることはない．

プロトタイプの開発終了をもってプロジェクトが終了する場合は，デバイスのコストはあまり問題にはならない．むしろ完成までに要する時間をいかに短縮するかで開発コストが左右されるので，開発ツールを含めていかに最適なプログラマブル・デバイスを選択するかがポイントとなってくる．

プロトタイプの開発に引き続き，量産モデルの開発に移行する場合は，1台あたりのコスト削減が大きなテーマとなってくる．この場合は，あらかじめハード・マクロへの移行が可能なデバイスを選択するとか，ゲート・アレイなどの量産を考えたプロトタイプの開発を行うということになる．

次に，同一回路を複数個必要とする場合，回路規模が膨らみすぎる場合がある．このような場合は，プログラマブル・デバイス内部に必要な数だけ回路を定義することで，回路規模の増大を抑えることができる．この手法はRS-232Cコントローラ，パルス・カウンタ，パルス・エンコーダ，PWM発生器などかなり広範囲にわたって利用できる．さらに，これらの回路の入力部分にオリジナルにはないディジタル・ノイズ・フィルタの追加などを行って，より付加価値を高めることも可能である．

リスト1にXilinx社のFPGAであるSpartan-IIシリーズであるXC2S100に4チャネルのノイズ・フィルタ付きパルス・エンコーダをコンフィグレーションしたときの結果を示す．

この結果からこのデバイスには最大36チャネルまでのパルス・エンコーダを組み込むことが可能ということがわかる．さらに多くのチャネルが必要な場合は，同シリーズのXC2S200を使用して同一パッケージで72チャネルまで組み込むことが可能である．

● 性能

図1に示すようにDSPは基本的にストアード・プログラム方式（ノイマン型ともいう）のアーキテクチャで動作するため，メモリ・アクセス時間や命令解釈など，本来信号処理と関係ない時間がプラスされる．非常に高速に信号処理を行う必要がある場合は，このような点も留意しなければならない．

リスト 1　XC2S100による 4チャネル・パルス・エンコーダのコンフィグレーション結果

```
Device utilization summary:
   Number of External GCLKIOBs          1 out of 4       25%
   Number of External IOBs             31 out of 92      33%
   Number of LOCed External IOBs        0 out of 31       0%
   Number of SLICEs                   143 out of 1200    11%
   Number of GCLKs                      1 out of 4       25%
   Number of TBUFs                     64 out of 1280     5%
```

表 1　Spartan-3(XC3S1000) で実現できる各機能の実行速度

Functions	Device Utilized	Key Specification
1024-point complex FFT	24.10%	20μsec
64-tap FIR filter	3.00%	8.1Msps
Digital up/down Converter	18.60%	100Msps
Viterbi decoder	37.80%	1.9Msps
Reed Solomon G.709 encoder	1.30%	120MHz
Reed Solomon G.709 decoder	6.90%	60MHz

表 2　各方法による同一機能のディジタル信号処理実装の得失(1 が最高順位)

	処理速度	コスト	規模	柔軟性	開発言語
PLD/CPLD	1	5	5	2	HDL
FPGA	2	4	4	1	HDL
ASSP	3	1	1	5	C
DSP	4	3	3	4	C
CPU	5	2	2	3	C

一方，PLD や FPGA などのプログラマブル・デバイスは，図 2に示すように同期回路で設計されているかぎり同期クロックの速度で演算が行われる．また，リソースが許すかぎり同一回路を複数個定義することで，高度なパラレル演算を行うことが可能である．これらによって表 1に示すように，FFT 演算をはじめとする各演算処理を DSP ではまねのできない速度で実行することが可能である．

表 2では，同一機能のディジタル信号処理を実現する際の各実装による得失について順番をつけてみた．上で述べたようにプログラマブル・デバイス，なかでも PLD と CPLD は処理速度で最高の順位となるが，コストの点では非常に高くつく．したがって，PLD と CPLD は一般には非常に小規模な回路に対して使用されることとなる．

さて，最新の FPGA ではプロセッサ機能も FPGA で実現することが可能となってきた．Xilinx 社では，MicroBlaze または PicoBlaze プロセッサ，ALTERA 社では NIOS プロセッサという名前で発表されている．これらのプロセッサは Linux などの汎用 OS がこの上で動作するので，ディジタル信号処理とホスト CPU 機能を同一の FPGA に収めることができる．組み込みシステムなどで非常に回路を小さくする必要がある場合は，図 3に示すように，FPGA 組み込みプロセッサと DSP 機能の組み合わせを検討してもよいだろう．

● 開発ツール

ディジタル信号処理実現のためのツールとしておもなものは，PLD，CPLD，FPGA については Project Navigator(Xilinx 社：図 4) や Quartus(Altera 社) がある．DSP については Code Composer Studio (TI 社) や VisualDSP++(Analog Devices 社) がある．PLD，CPLD，FPGA については VHDL や Verilog-HDL といった HDL，DSP については C またはアセンブラで開発を行う．これらの開発ツールは

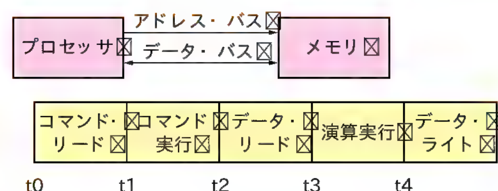


図 1　ストアード・プログラム・アーキテクチャ

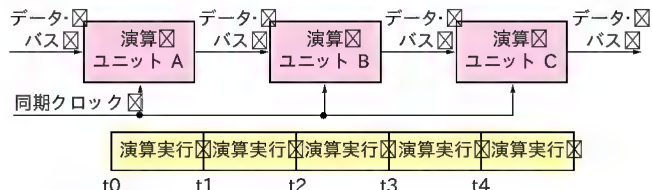


図 2　同期式パイプライン演算処理

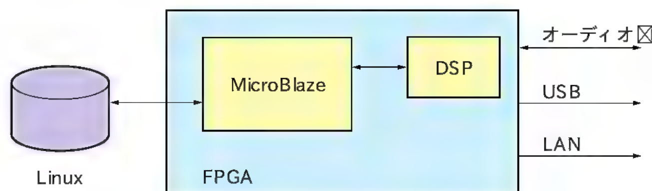


図 3　FPGA 組み込みプロセッサのブロック図

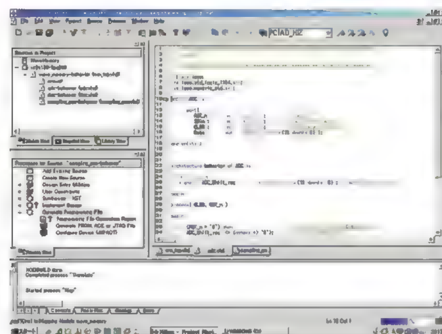


図 4
Xilinx 社の Project Navigator

Windows や Linux といった汎用 OS で動作し，プログラムのプロトタイピングから最終的なデバッグ，ROM 作成までの統合開発環境を提供する．

まとめ

ディジタル信号処理を実現するための方法は上に示したようにいくつもの方法がある．どの方法を選ぶにしても，デバイスやツールの理解が十分でなければ，正しい見通しを得ることが難しくなる．また，デバイスやツールはまさに日進月歩の勢いで進化を遂げている．そのため，日ごろから最新の情報について研究を怠らないようにしなければならない．

しかし，何よりも重要なことは，ともすればデバイスやツールの細かい情報に振り回され，それだけに心を奪われがちだが，バックボーンとしてディジタル信号処理の本質の理解があってこそ，それらが意味を持つということを忘れてはならない．

おおひさ・のぶひろ　ドリームコンピュータ(株)

関連記事 特集

MATLABプロダクト・ファミリがバージョン・アップ

MATLAB Release 14の強化機能と追加機能

柴田 克久

ディジタル信号処理システム開発をはじめ，広範な産業分野において利用されているMATLABプロダクト・ファミリがバージョン・アップし，2004年6月4日からサイバネットシステム(株)より国内向けの販売が開始されました．

最新バージョンRelease 14は，2002年のRelease 13以来，2年ぶりのメジャー・バージョン・アップとなり，基本ツールであるMATLABとSimulinkの機能拡張に加え，各種のオプション・ツール(Toolbox，Blockset)が新製品として追加されています．ここでは，今回のバージョン・アップのトピックを紹介していきます．

● バージョン・アップの概要

Release 14には，各種の産業や適用分野向けの12種類の新規オプション・ツールが追加されています．また，既存の28種類のモジュールにおいて機能を拡張し，30以上の既存製品において，マイナ・バージョン・アップが実施されています(表1)．

MATLABプロダクト・ファミリの開発元である米国The MathWorks社では，今回のバージョン・アップに関して以下の2点を最重要項目として挙げています．

▶ データ処理速度と解析の性能向上 —— MATLAB7

Release 14の基本モジュールであるMATLAB7では，プログラミングの生産性を向上させる各種の機能が追加されました．従来の倍精度演算に加え，整数や単精度演算といったデータ・タイプのサポートにより，従来と比較して，大規模なデータ・セットをより高速に処理することができるなど，性能が大幅に強化されています．さらに計算・解析のパフォーマンス改善のために，データ・タイプ，演算子，関数にわたる内部機能において多数の最適化が加えられています．

▶ 大規模で複雑な開発作業を効率化 —— Simulink 6

近年の通信システムに代表される大規模かつ複雑なリアルタイム組み込みシステムの開発においては，各工程間でモデルを共有しながら共同作業を進めていく新しい開発手法として「モデル・ベース設計」が提唱されています．Simulink 6には，このモデル・ベース設計をサポートするさまざまな機能が追加されています．たとえば，新機能であるModel Referenceが提供するコンポーネント・ベースのモデリングと，新しいGUI環境であるModel Explorerが提供する統合されたデータ管理機能が挙げられます．この機能により設計チームは，組織内あるいは複数の組織に渡って，従来は管理が困難であったシステム全体の設定やパラメータを共有し，サブシステム上での個別の開発作業や，協調作業を効率的に行うことができます．

● 信号処理・通信システム開発用ツールの新機能

信号処理・通信システム開発の分野においては，主としてMATLAB/Simulinkと組み合わせて利用するオプション・ツールの拡充と，既存ツールへの機能追加がなされています．特にフィルタ設計，開発における実装を意識した機能強化と，従来は対応できなかった新しい適用分野への対応が進められています．

● 固定小数点機能の強化

▶ Fixed-Point Toolbox

MATLABに固定小数点データ・タイプと計算機能を追加するオプション・ライブラリです．これによりフィルタ設計などの分野において，固定小数点によるアルゴリズム開発のテストやモデリング，システムの実現検証用の関数，演算子が利用できます．

▶ Simulink Fixed Point

このオプションにより，Simulinkプロダクト・ファミリのもつ固定小数点機能が利用可能となります．従来はSimulink上で固定小数点モデルを作成する場合，ブロックの置き換えが必要となっていましたが，Release 14ではその必要がなくなり，ブロックを変更することなく，固定小数点モデルに変換することができるようになりました．

● フィルタ設計・実装機能の強化

▶ Filter Design Toolbox

従来のFIR，IIRフィルタの設計機能に加え，2次(Second-Order Section：SOS)フィルタのカスケード接続によるIIRフィルタを直接設計できるようになりました．これにより，より実装を意識したフィルタ設計が可能となり，先に述べたFixed-Point Toolboxと組み合わせて，固定小数点フィルタ出力の量子化誤差を確認しながら設計することができます．さらに適応フィルタ，マルチレート・フィルタの設計機能も強化されています．

▶ Filter Design HDL Coder

このツールにより，Filter Design Toolboxで設計された固定小数点フィルタの効率的で合成可能なVHDL，またはVerilog

表1　MATLABプロダクト・ファミリ Release14のバージョン・アップの内容

メジャー・バージョン・アップされたオプション製品(28種類)	
Communications Blockset	Communications Toolbox
Database Toolbox	DSP Blockset (Signal Processing Blockset)
Embedded Target for Motorola MPC555	Embedded Target for TI C6000 DSP
Filter Design Toolbox	Financial Derivatives Toolbox
Fixed-Point Blockset (Simulink Fixed Point)	Instrument Control Toolbox
Mapping Toolbox	MATLAB Compiler
MATLAB Report Generator	Model Predictive Control Toolbox
Nonlinear Control Design Blockset (Simulink Response Optimization)	Optimization Toolbox
Real-Time Workshop	Real-Time Workshop Embedded Coder
Signal Processing Blockset	Simulink Fixed Point
Simulink Report Generator	Simulink Response Optimization
Stateflow	Stateflow Coder
Statistics Toolbox	System Identification Toolbox
Virtual Reality Toolbox	Wavelet Toolbox
オプション・ツールの新製品(12種類)	
Bioinformatics Toolbox	Embedded Target for TI C2000 DSP
Filter Design HDL Coder	Fixed-Point Toolbox
Genetic Algorithm and Direct Search Toolbox	Link for ModelSim
OPC Toolbox	RF Blockset
RF Toolbox	Simulink Control Design
Simulink Parameter Estimation	Simulink Verification and Validation
その他アップデートされた製品	
Aerospace Blockset 1.6	Data Acquisition Toolbox 2.3
Datafeed Toolbox 1.5	Dials & Gauges Blockset 1.2
Embedded Target for Infineon C166 Microcontrollers 1.1	
Embedded Target for Motorola HC12 1.1	Embedded Target for OSEK/VDX 1.1
Excel Link 2.2	Extended Symbolic Math Toolbox 3.1
Financial Time Series Toolbox 2.1	Financial Toolbox 2.4
Fixed-Income Toolbox 2.1	Fuzzy Logic Toolbox 2.1.3
GARCH Toolbox 2.0.1	Image Acquisition Toolbox 1.5
MATLAB Link for Code Composer Studio® 1.3.1	
MATLAB Builder for COM 1.1(MATLAB COM Builderから名称変更)	
MATLAB Builder for Excel 1.2(MATLAB Excel Builderから名称変更)	
Model-Based Calibration Toolbox 2.1	
Neural Network Toolbox 4.0.3	
Partial Differential Equation Toolbox 1.0.5	Real-Time Windows Target 2.5
Signal Processing Toolbox 6.2	SimMechanics 2.2
SimPowerSystems 3.1	Simulink Accelerator 6
Symbolic Math Toolbox 3.1	xPC Target 2.5
xPC Target Embedded Option 2.5	xPC TargetBox 2.5

*青文字は今回紹介しているツール.

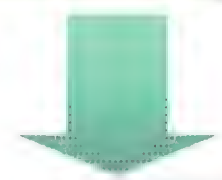
のコードを自動生成し，ASICやFPGAをターゲットとして実装することができます．生成されたコードは，ベンダに依存しない汎用的で可読性の高いコーディング・スタイルで記述されており，容易にカスタマイズを行うことが可能です．また，検証用にVHDLもしくはVerilogのテスト・ベンチを生成することも可能です(図1)．

▶ Link for ModelSim

FPGAおよびASIC開発のためのハードウェア設計フローへMATLAB，Simulinkを統合するコシミュレーション・インターフェースです．これは，MATLAB/SimulinkとMentor Graphics社のModelSim(HDLシミュレータ)との間に高速な双方向リンクを提供します．

▶ Embedded Target for TI C6000 DSP

従来のTI社製C67x浮動小数点DSPおよびC62x固定小数点DSPに加え，新たにC64xシリーズの浮動小数点DSPをサポートするようになりました．



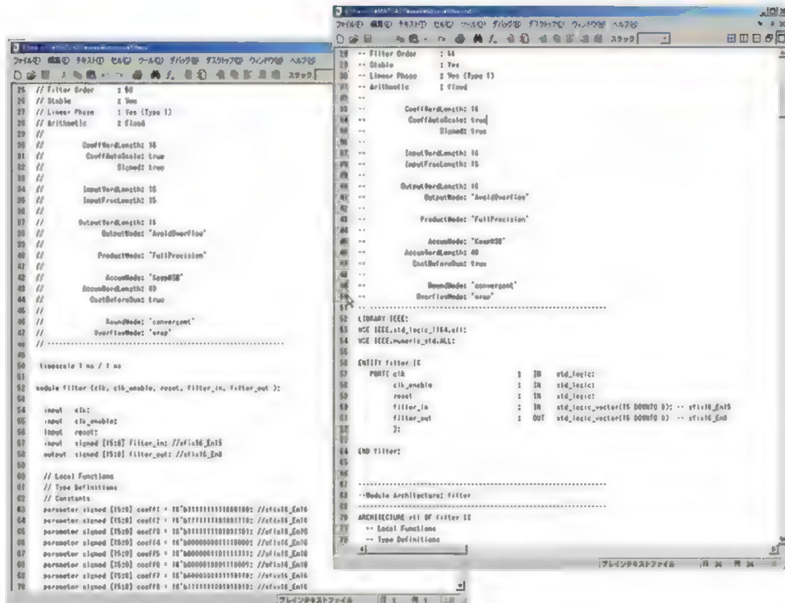


図1　FDAToolによるフィルタ設計・HDL生成——GUI環境でフィルタ設計，量子化効果の確認，VHDL/Verilogコードの生成が可能

● 通信システム開発用の新製品と新機能

▶ Communications Toolbox

従来から，通信システム開発向けのMATLABオプション・ライブラリとして，通信向けのアルゴリズム関数や専用のプロット・ツールを提供していましたが，今回のバージョン・アップで，マルチパス・チャネル・モデル，インタリーバ，イコライザ，特性評価関数を含む，通信リンクの物理層の設計のための42の新しい関数が追加されました．またBERToolという新しいGUIにより，設計した通信システムの特性評価を容易に実現することができます．

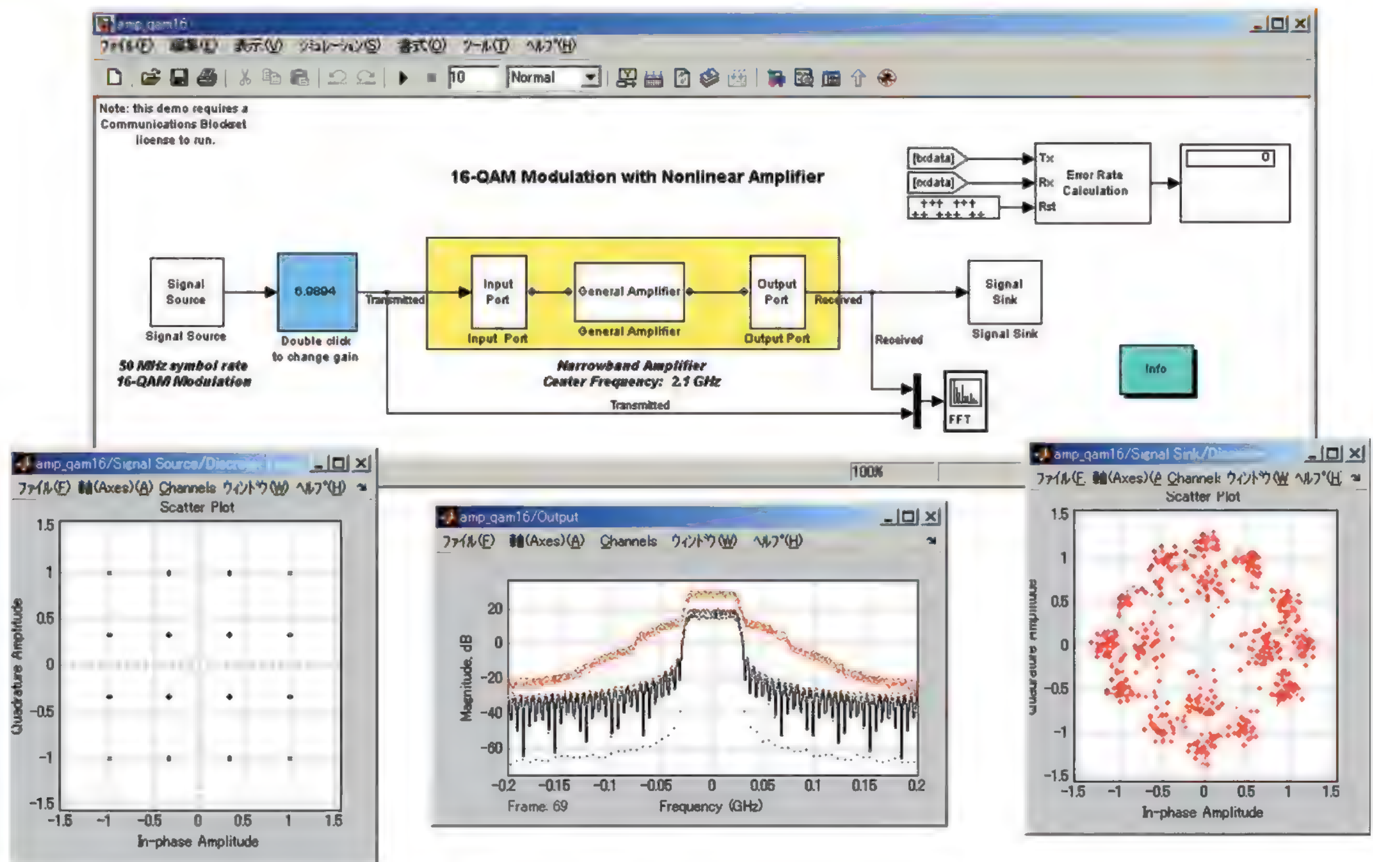


図2　RFアンプを含む16QAM通信システム・シミュレーション
RFアンプの非線形特性による影響をシミュレーションで確認することができる

▶ Communications Blockset

通信システムの物理レイア設計向けのSimulinkブロック・ライブラリに，新たに適応等化器や同期のブロックが追加されました．これにより，より現実に即した通信システムの受信器の設計と開発が可能となりました．

▶ RF Toolbox

新しいMATLABのオプション・ライブラリで，これによりアンプやフィルタ，ミキサや伝送線路などのようなRFコンポーネントの利用が可能となります．各コンポーネントの特性は，ネットワーク・パラメータ(S，Y，Z，hなど)や数学的な挙動，物理的なプロパティにより指定され，それぞれを接続した場合の特性を算出することも可能です．また，業界標準のファイル・フォーマットをサポートし，ほかのツールのネットワーク・パラメータ・データを読み込むこともできます．またスミス・チャートや極座標など RF 解析専用のプロット機能も提供します．

▶ RF Blockset

Simulinkのオプション・ライブラリとして，RFコンポーネントの挙動をモデル化するためのブロックを提供します．RF Toolboxで作成されたRFコンポーネントをSimulink環境上のブロックとして取り込むことにより，RF劣化などの非線形特性がシステム性能に与える影響をシミュレーションすることが可能となります(図2)．

しばた・かつひさ　サイバネットシステム(株)

TECH I Vol.9　好評発売中
シミュレーションで学ぶディジタル信号処理
MATLABによる例題を使って身につける基礎から応用
B5判164ページ
尾知 博 著
定価2,000円(税込)
ISBN4-7898-3320-8
CQ出版社　〒170-8461 東京都豊島区巣鴨1-14-2　販売部 TEL.03-5395-2141　振替 00100-7-10665

関連記事 特集

AccelChip による FPGA 設計

ディジタル信号処理アルゴリズムのハードウェア化手法

持田 雅行

ディジタル信号処理をハードウェア上で実現することへの障害

従来，ディジタル信号処理を行うには，DSP を使用し，プログラムによって実現することが一般的だった．近年，信号処理速度の高速化，低消費電力化などの要求とともに，ディジタル信号処理アルゴリズムを FPGA などのハードウェアで実現させるという要求が高まってきている．

ディジタル信号処理アルゴリズムの開発では，多くの技術者が MathWorks 社の MATLAB を使用している．MATLAB は短いプログラム・ステップで目的とするアルゴリズムを効率よく記述することができるため，アルゴリズムの開発検証には非常に有効なツールである．しかし，MATLAB で開発されたアルゴリズムを FPGA 上で実現するためには多くの困難な作業が伴うため，ディジタル信号処理に FPGA はあまり使われていなかった．

MATLAB のアルゴリズムをそのまま実装すると浮動小数点演算が使われるが，ハードウェア上で実現するためには固定小数点でアルゴリズムを書き直す必要がある．使用されている各変数に対して，量子化関数を使用して最適なビット幅を指定し，アルゴリズムが正しく変換されているかどうかを検証していく作業は手間と時間がかかる．また，アルゴリズムを固定小数点モデルで実現できたとしても，ハードウェアを実現するために VHDL や Verilog などの RTL でプログラムすることは，アルゴリズム・エンジニアと，ハードウェア・エンジニアの間で仕様確定や，プログラミングのために多くの時間が費やされてしまう．また RTL モデル化の段階で，アルゴリズムの修正が必要な事態が発生した場合は，アルゴリズムの開発，固定小数点モデルの作成，RTL モデルの作成などすべてを再設計しなくてはならない．このようなことが，ディジタル信号処理をハードウェアで実現することの障害となってきていた．

AccelChip DSP Synthesis ツールの登場

MATLAB の M ファイルから論理合成可能な RTL モデルを自動生成するツールとして，AccelChip 社は，AccelChip DSP Synthesis というソフトウェアをリリースした．

この技術は，ノースウェスタン大学の Prith Banerjee 教授が，1998 年度に米国国防総省高等研究事業局の適応演算システム・プロジェクトの補助金により開発した技術がベースになっている．AccelChip 社はこの技術の商業化の権利を得て，2002 年に AccelFPGA という FPGA に特化したツールを最初の製品としてリリースした．

その後，製品に改良が加えられ，2004 年に AccelChip DSP Synthesis と名を改め，FPGA，ASIC，ストラクチャード ASIC などに対応した RTL 生成ツールとして登場した．

この製品では，AccelWare という，MATLAB の Communication Tool Box，Signal Processing Tool Box に対応した関数のライブラリがサポートされている．Tool Box 関数を直接使用したものは，そのままでは論理合成可能な RTL に変換ができないため，AccelWare で置き換えることにより，論理合成が可能になる．

このツールがリリースされたことにより，MATLAB の M ファイルから FPGA を設計するトップダウンの設計フローが提供されることになった．従来，時間がかかっていた手作業によるプロセスが AccelChip を導入することで，アルゴリズム設計から FPGA に実装するまでの時間を大きく短縮することが可能になった．

AccelChip による設計プロセス

AccelChip DSP Synthesis は，MATLAB の M ファイルから論理合成可能な RTL を生成する設計ツールである．AccelChip を使用した設計プロセスのフローを図 1 に示す．

（1）M ファイルによるアルゴリズム設計（図 1①）

信号処理アルゴリズムを MATLAB の M ファイル形式で記述する．この場合，AccelChip のガイドラインに従った記述が必要となる．具体的には，FPGA 化するアルゴリズムを記述した M ファイル（ファンクション M ファイル）と，このファンクション（関数）に入力するデータを生成したり，出力されたデータを評価するプロセスを記述した M ファイル（スクリプト M

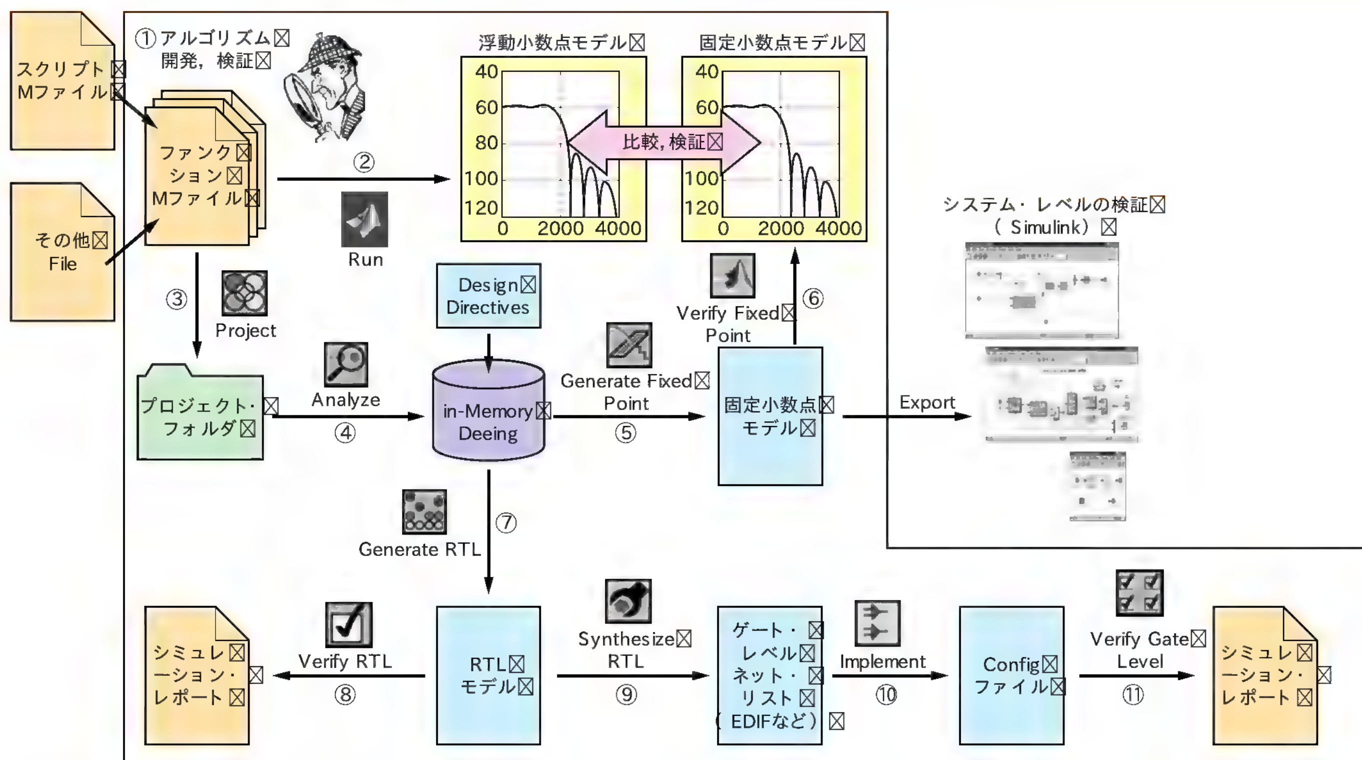


図1　AccelChipによるデザイン・フロー

ファイル）の二つが最低限必要である．ファンクションMファイルは，スクリプトMファイル内の，ストリーミング・ループと呼ばれる`for`ループ内より呼び出される必要がある．

設計したディジタル信号処理のアルゴリズムはMATLABの環境下でアルゴリズムの正当性を検証する．検証が終わったMファイルは，全設計プロセスで変更されることのないゴールデン・ソースとして位置付けられる（図1②）．

（2）AccelChipの起動/プロジェクト設定（図1③）

AccelChip DSP Synthesisを起動し，最初に行うのはプロジェクトの設定である．“Project”アイコンにより設定プロセスに入る．初めて設計を行うときは，作業フォルダを指定し，フォルダ内にプロジェクト・ファイルを設定する．プロジェクトは拡張子が`.acc`というファイル名で保存される．プロジェクトのオプションとして，ターゲットとするFPGAのベンダ，使用デバイス名，クロック周波数，生成するRTL（VHDL，Verilog），使用するシミュレーション・ツール，論理合成ツール，配置配線ツールなどを指定することができる．

（3）Mファイルの構造解析（図1④）

プロジェクトを設定した後，“Analyze”アイコンにより，AccelChipはMファイルの構造がAccelChipの設計ルールに従っているかどうかを解析する．設計の構造が正しければ，スクリプトMファイル内のどの部分をFPGAとして設計するかの設定入力を要求する．複数のストリーミング・ループが存在するときは，そのうちの一つのみが，設計の対象となる．

（4）固定小数点モデルの自動生成と検証（図1⑤）

設計対象を決定すると，固定小数点モデル生成のアイコン（Generate Fixed Point）が表示される．固定小数点モデルへの変換はこのアイコンをクリックするだけで自動的に行われる．レポート画面上には固定小数点モデル生成時のhtml形式の結果レポートが表示される．このレポートにより，ファンクション・ブロック内の各変数に対し，どのような量子化が行われたかを知ることができる．

固定小数点モデルが生成されると，固定小数点モデルの検証アイコン（Verify Fixed Point）が，新しく表示される．このアイコンをクリックすると，生成された固定小数点モデルを使用してMATLABが起動される（図1⑥）．

固定小数点モデルは，浮動小数点モデルのMファイルに量子化関数を使用しただけのものであり，浮動小数点モデルでプログラムしたものと同じ出力結果が表示される．この出力結果と浮動小数点モデルの出力結果を比較検討することで，固定小数点モデルが設計アルゴリズムを満足するものであるかどうかを検証する．もし，オーバフローやアンダフローの発生などにより，生成された結果が満足できるものでなかったら，固定小数点モデル生成時の結果レポート上に表示されている変数を画面上でクリックすることで，ビット幅，フラクション・ビット幅などの量子化のパラメータを簡単に変更することができる．変更した内容は，デザイン・ディレクティブ・ファイルとして拡張子`add`のファイルに保存される．

リスト 1　スクリプト Mファイル

```
% 輪郭抽出　画像処理　スクリプト・ファイル (edgedete_script.m)

image_in    = imread('flower1.jpg');        % 画像データ読み込み

%!ACCEL SYNTHESIS OFF
data_s      = double(image_in(:,:,3));      % 実数形式に変換
%!ACCEL SYNTHESIS ON

num_rows = 320;
num_cols = 250;

NUM_ITERATIONS = num_cols*num_rows;         % 入力データ総数

indata          = reshape((data_s'), 1, num_rows*num_cols);

% 入力データ表示
figure; imshow(uint8(data_s)); colormap(gray);title('Input to Edge detector');

% ストリーミング・ループ
 for n = 1:NUM_ITERATIONS
        indatabuf    = indata(n);                          % 入力バッファ
        outdatabuf   = accel_edge1(indatabuf, num_cols);   % デザイン・ファンクション
        outdata(n)   = outdatabuf;                         % 出力バッファ
 end

 % 出力結果の表示
 outdata_tmp = reshape(outdata,num_cols,num_rows);
 figure; imshow(uint8(outdata_tmp')); colormap(gray);title('Result from Floating point Simulation');
```

(5) RTLモデルの生成(図1⑦)

固定小数点モデルの設計が確定したら，RTLモデルへの変換は，RTLモデル生成用のアイコン(Generate RTL)をクリックするだけで自動的に行える．生成結果のレポートとして，デザイン・ファンクション内で使用される積算器，加算器，減算器の使用状況，およびAccelChipが自動的に付加した入出力インターフェース部の信号名などが表示される．

(6) RTLモデルの検証(図1⑧)

RTLモデル検証用のアイコン(Verify RTL)をクリックすることで，プロジェクト・オプションで指定したシミュレーション・ツールが実行される．ModelSim，Rivieraがサポートされている．使用するテスト・ベンチは，最初のスクリプトMファイルから，固定小数点モデルの検証時に自動生成されたものが使用される．

(7) 論理合成(図1⑨)

RTLモデルの検証がパスしたら，論理合成(Synthesize)のアイコンをクリックすることで論理合成用ツールを呼び出すことができる．Synplify，Synplify Pro，XST，QuatusⅡがサポートされており，プロジェクト・オプションで選択することができる．論理合成結果として，各ベンダ対応のネット・ファイル，あるいは標準的なEDIFが出力される．

(8) 配置配線(図1⑩)

論理合成が終わったら，配置配線のアイコン(Implement)により，ISE，QuatusⅡなど，プロジェクト・オプションで指定したツールを起動することができる．このステップで，ターゲットFPGA用のコンフィグレーション・ファイル，およびゲート・レベルのシミュレーション用ファイルが生成される．

(9) ゲート・レベル検証(図1⑪)

ゲート・レベル検証用のアイコン(Verify Gate Level)をクリックすることで，ゲート・レベルのシミュレーションを行うことができる．ここで使用される入力データは，固定小数点モデルの検証を行ったデータと同じものが使用される．ここで検証がパスすれば，できあがったFPGAは，浮動小数点モデルのMATLABに対して，Bit-trueであることが保証される．



設計例――画像処理：輪郭抽出――

それでは，AccelChipを使用した設計の流れを，ファイルより画像データを取り込み，輪郭を抽出して表示するという，簡単な画像処理のアルゴリズムを例に取って説明する．

● アルゴリズムの設計と検証

リスト1に輪郭抽出用のスクリプトMファイルを示す．対象とする画像データは，ファイルより読み込み，reshapeを使用して，1次元データ“indata”に変換している．この入力データ列は，テスト・ベンチ用データとしても使用される．

forループは，ストリーミング・ループと呼ばれ，設計対象である輪郭抽出アルゴリズムのaccel_edge1を呼び出している．デザイン・ファンクションへの入出力である入力データ・ストリーム“indata”，出力データ・ストリーム“outdata”は，indatabuf，edge_pixelでデザイン・ファンクションとインターフェースされている．

これらのバッファ部は，ハードウェア上で入出力バッファとして実現される．

リスト2に，設計対象となるファンクションMファイル“accel_edge1”を示す．ファンクション内ではスクリプト・

リスト 2　ファンクション M ファイル（accel_edge1.m）

```
function edge_pixel  = accel_edge1(indatabuf, num_cols);
   persistent row_buf;
   if isempty(row_buf) row_buf = zeros(1, num_cols); end;

   indatabuf_char = indatabuf;
   threshold    = 8;

   % ひとつ上のピクセルと比較
   y_diff = abs(row_buf(num_cols) - indatabuf_char);

   % ひとつ左のピクセルと比較
   x_diff = abs(indatabuf_char- row_buf(1));

   %各ピクセル間の差をスレッショルドと比較
   if (y_diff > threshold)
      edge_pixel_char = 0;
   elseif (x_diff > threshold)
      edge_pixel_char = 0;
   else edge_pixel_char = 255;
   end
  % ライン・バッファを更新
   row_buf = [indatabuf_char row_buf(1:end - 1)];
   edge_pixel = edge_pixel_char;
```



図 2　原画像

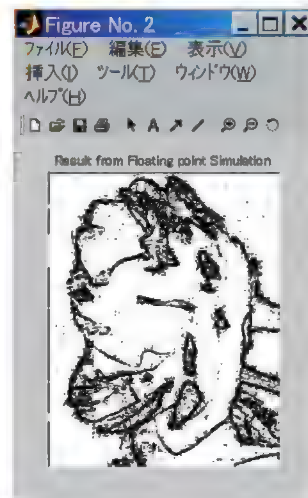


図 3　輪郭抽出結果

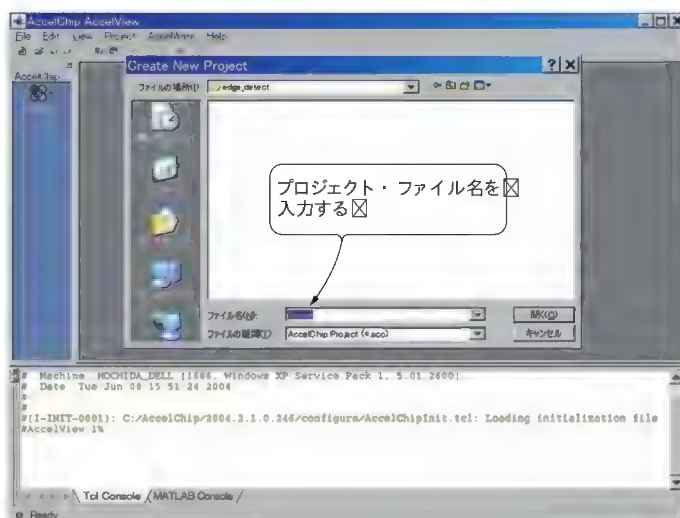


図 4　プロジェクトの設定

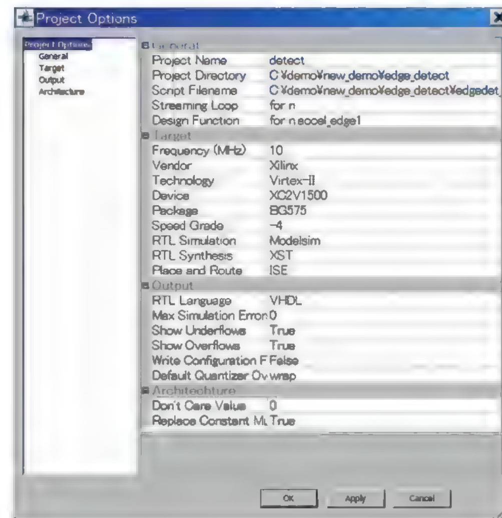


図 5　プロジェクト・オプションの設定

ファイルより引き渡されたピクセル・データを，1 行上のピクセル，および一つ左のピクセルと比較を行い，絶対値がスレッショルド値より大きい場合はデータ値 255 を，小さい場合は 0 を設定している．このようにすることで画像の輪郭を取り出している．persistent で規定した row_buf には，入力データを順次挿入していくことで，データを比較するための 1 行上のピクセルと一つ左のピクセルのデータを保存しているライン・バッファを形成している．

浮動小数点モデルで実行した結果を図 2，図 3 に示す．図 2 が原画像で，図 3 が輪郭抽出した画像である．

この出力結果が，固定小数点モデルの結果を比較する対象となる．

● プロジェクト設定

AccelChip を起動し，“Project”アイコンをクリックすると，図 4 のようにプロジェクト・ファイルの入力を要求される．スクリプト M ファイル，ファンクション M ファイルなどのデザイン・ファイルがあるフォルダ内にプロジェクトを指定する．このフォルダ内には設計フロー内で生成される各種出力ファイル，レポート・ファイルなどが保存されていく．

次に，“Project Option”として，FPGA ベンダ名，使用デバイス，出力する RTL の種類，シミュレーション，論理合成用の各種ツールを設定する（図 5）．

“Analyze”アイコンをクリックすると，図 6 で示すように，プロジェクトで使用するスクリプト M ファイルの指定を要求される．この場合 edgedetect_script を指定する．続いて，スクリプト・ファイル内のストリーミング・ループ構造が図 7 のように表示されるので，設計対象の accel_edge1 を選択する．

● 固定小数点モデルの生成

ここまで設定が終われば，“Generate Fixed Point”アイコンをクリックするだけで，固定小数点モデルの自動生成が行われ

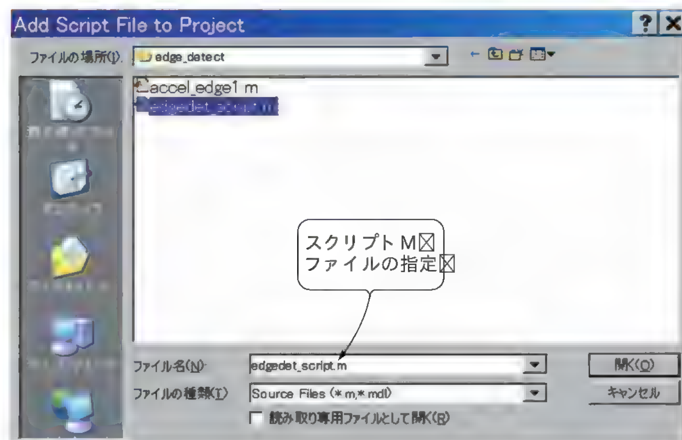


図6　スクリプト・ファイルの選定

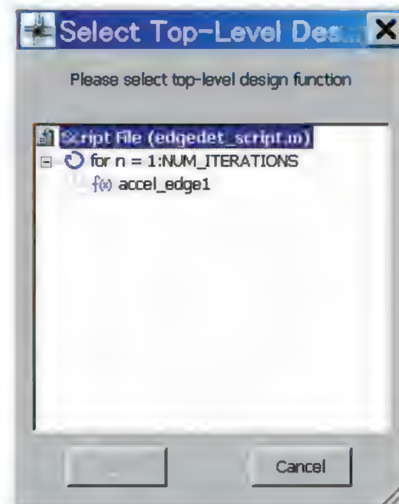


図7　デザイン・ファンクションの指定

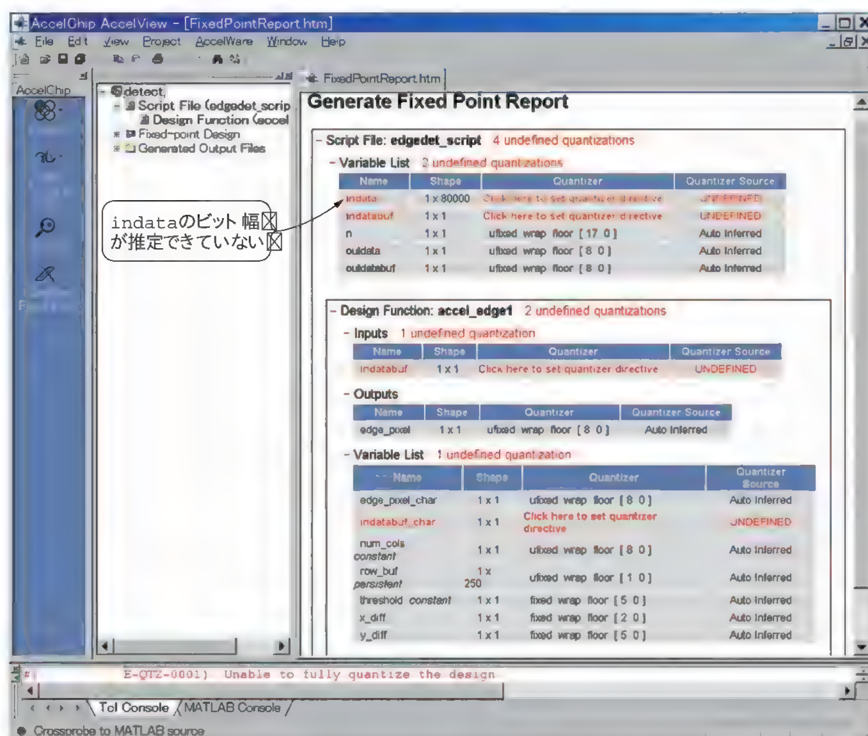


図8　固定小数点モデル生成レポート

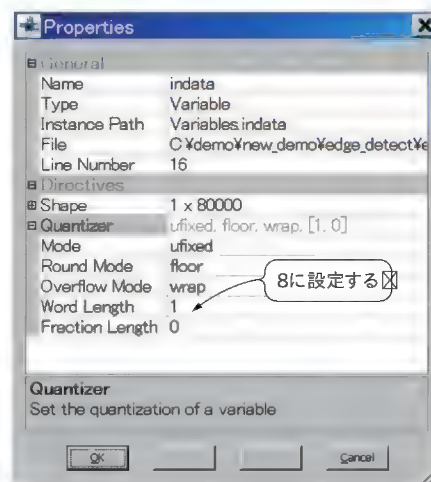


図9　indataのプロパティ設定

る．固定小数点モデルが生成されると，html形式のレポートが，図8のように表示される．

この例では，入力データであるindataに関して，AccelChipは正しく判断が行えないことを通知している．

これは，indataは，ファイルからロードしてくるために，実際のデータがロードされるまで，そのダイナミック・レンジを判断することができないためである．

この場合，入力データ列の最大値は255なので，ビット幅として8ビットをAccelChipツールに教える必要がある．固定小数点モデル生成レポート内で，indataのQuantizer欄をクリックすると，indataに関するプロパティ設定画面が図9のように表示される．この中で“Word Length”の設定を8に設定する．変数のプロパティを変更すると，その変更結果はすぐに固定小数点モデル全体に反映され，変更が影響する変数すべてに対して自動的に量子化の結果が反映される（図10）．

● 固定小数点モデルの検証

この状態で，“Verify Fixed Point”アイコンをクリックすると，MATLABが起動し，固定小数点モデルのスクリプト・ファイルを実行し，図11のような結果が得られる．

この結果を浮動小数点モデルの結果（図3）と比較すると，出力結果は期待値よりずれており，固定小数点モデルとしてまだ問題があることがわかる．

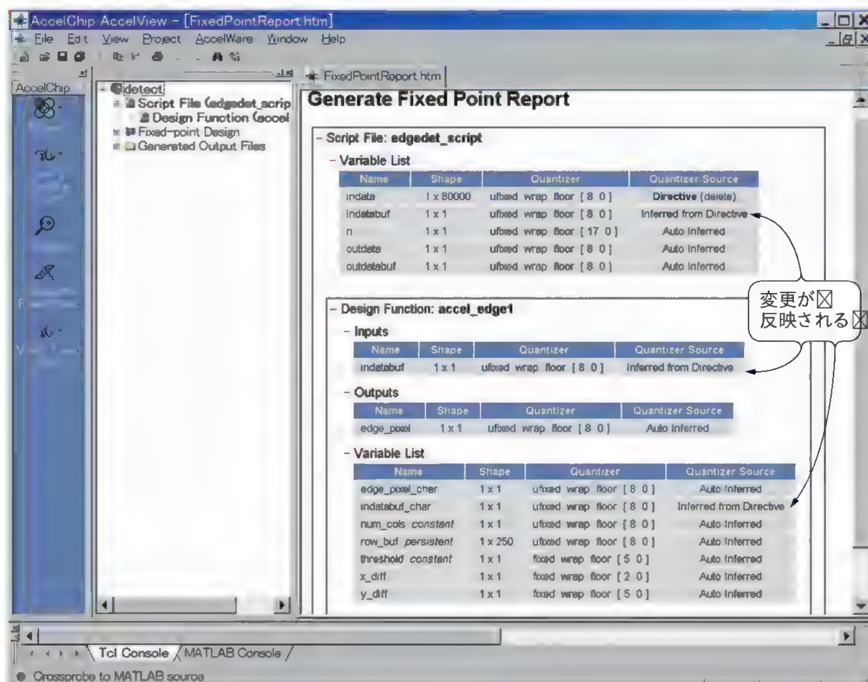


図10　他の関数へも変更が反映する

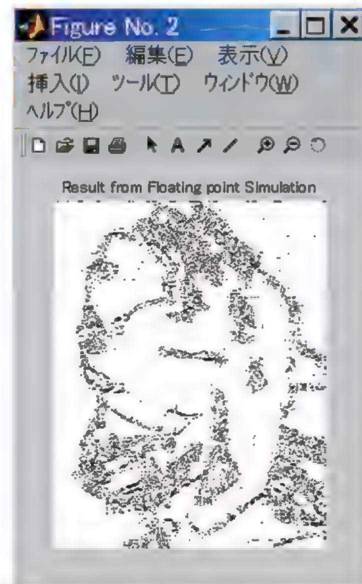


図11　固定小数点モデルの出力結果-1

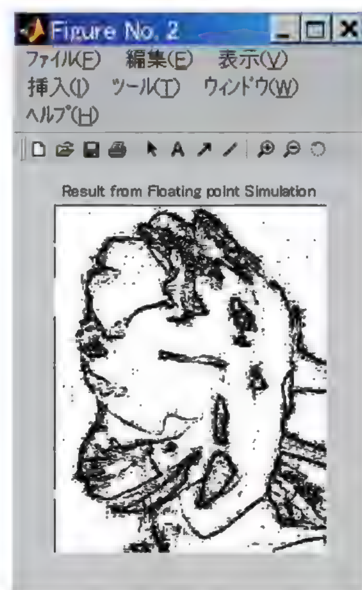


図12　固定小数点モデル出力結果-2

リスト3　固定小数点モデル(一部)

```
function edge_pixel = accel_edge1 ( indatabuf )
num_cols = quantize( quantizer('ufixed','floor','wrap',[ 8, 0 ]), 250 );
    :
    :
threshold = quantize( quantizer('fixed','floor','wrap',[ 8, 0 ]), 8 );
% ひとつ上のピクセルと比較
y_diff = quantize( quantizer('fixed','floor','wrap',[ 9, 0 ]),
abs( quantize( quantizer('ufixed','floor','wrap',[ 8, 0 ]),
row_buf(quantize( quantizer('ufixed','floor','wrap',[ 8, 0 ]), num_cols )) ) -
quantize( quantizer('ufixed','floor','wrap',[ 8, 0 ]), indatabuf_char ) ) );
% ひとつ左のピクセルと比較
x_diff = quantize( quantizer('fixed','floor','wrap',[ 9, 0 ]),
abs( quantize( quantizer('ufixed','floor','wrap',[ 8, 0 ]), indatabuf_char ) -
quantize( quantizer('ufixed','floor','wrap',[ 8, 0 ]), row_buf(1) ) ) );
    :
    :
```

固定小数点レポート(図10)を改めてよく見てみると，x_diffで，ビット幅が2ビット，y_diffで5ビットしかとられていないことがわかる．これらの値は，ピクセル値の差をとるのでその絶対値の最大値は255となり，そのためには最低8ビットのデータ幅を必要とするはずである．AccelChipの自動量子化の過程では，ビット幅を推定するのにループを1回しかみておらず，すべての取りうる値をチェックしていないことに起因している．このような変数に対しては，直接ビット幅を指定する必要がある．

レポート内のx_diff，y_diffのQuantizerの欄をクリックし，両変数のプロパティ設定で，"Word Length"を符号ビットも考慮してビット幅を9に設定する．

再度，固定小数点モデルの検証を行うと，図12の結果が得られる．この結果は浮動小数点モデルと同等の結果となり，固定小数点モデルの検証はここで完結する．固定小数点モデルに変換されたMファイルをリスト3に示す．

リストからわかるように，量子化関数が各変数に追加されている．

● RTLモデルの生成，検証

次のステップは，固定小数点モデルを基にした，RTLモデルの自動生成となる．"Generate RTL"アイコンをクリックすると自動的にRTLモデルが生成される．RTLモデル生成のレポートとして，オプションで設定したターゲット・デバイスの情報，積算器，加算器，減算器などの使用状態，AccelChipが提供する外部とのインターフェース信号などが図13のように

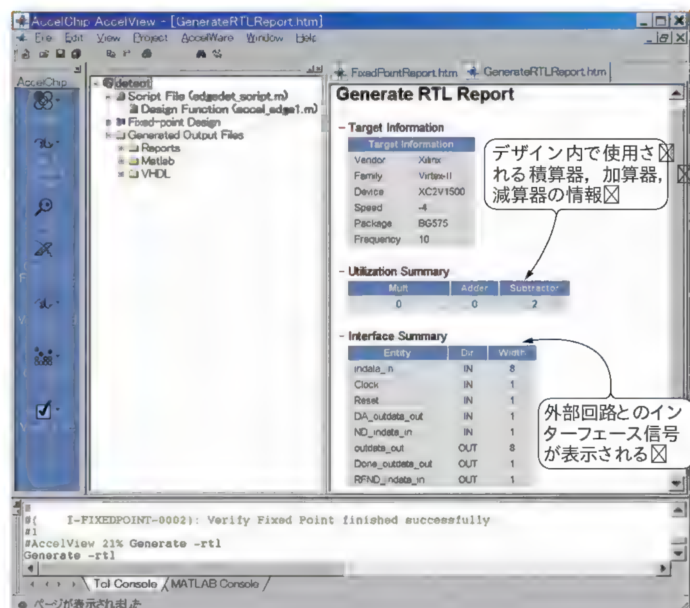


図 13　RTLモデル生成レポート

リスト 4　RTLモデル(一部)

```
indatabuf_char := indatabuf;
ac_tmp_0 := (signed(accel_resize(ac_scalar_row_buf_250_1,
           9)) - signed(accel_resize(indatabuf_char, 9)));
if (proceed_var0) then
    mf_0 := (ac_tmp_0(8) = '1');
    notmf_0 := ( not mf_0 );
else
    mf_0 := false;
    notmf_0 := false;
end if;
if (notmf_0) then
        y_diff_1 := ac_tmp_0;
else
    if (mf_0) then
        y_diff_1 := ("000000000" - ac_tmp_0);
    else
        y_diff_1 := (others => '0');
    end if;
end if;
if (proceed_var0) then
    mf_3 := (y_diff_1 > signed(accel_resize(threshold, 9)));
else
    mf_3 := false;
end if;
```

示される．生成されたRTLモデルの一部をリスト4に示す．

RTLモデルが生成されると，"Verify RTL"アイコンをクリックすることで，RTLレベルでの検証が行える．オプションで指定したRTLシミュレーション・ツールが起動され検証を行う．結果はコンソール・ウィンドウ上に示される．

● 論理合成，配置配線，ゲートレベル検証

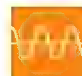
生成されたRTLモデルは"Synthesize RTL"アイコンをクリックすることで，オプションで指定した論理合成ツールにより論理合成される．出力されるネット・リストは，ベンダ特有のもの，またはEDIFのような汎用ファイルとなる．

"Implement"アイコンをクリックすることで，オプションで指定したツールが起動し，生成されたネット・ファイルを使用し，配置配線を行い，ターゲット・デバイス用のコンフィグレーション・ファイルを生成する．このとき，ゲート・レベルでのシミュレーション用ファイルがRTL(Verilog/VHDL)で生成される．

"Verify Gate Level"アイコンをクリックすることで，前段階で生成されたシミュレーション用ファイルを用いて，オプションで指定したシミュレーション・ツールを起動し，ゲート・レベルの検証が行われる．

以上，見てきたように，多くのステップは設計フローのアイコンをクリックしていくだけで，ほぼ自動的に目的とするRTLモデルを生成することができる．設計フローの各ポイントで設計検証が行えるため，問題のある部分は各ステップ内で解決していくことができ，最初のアルゴリズムまで戻って再設計する必要はない．

各ステップで生成されたファイルはプロジェクトを設定したフォルダ内で作成された独自のフォルダ内に保存されるため，いつでもほかのツールで使用することができる．

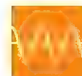


ハードウェア上での実現

AccelChipを用いて生成された結果がハードウェア上でどのように実現されているのかを説明する．MATLABスクリプトMファイルで表現されたストリーミング・ループは，実際のハードウェア上と，図14のように対応される．各入出力バッファは，外部のハードウェアとAccelChipの標準ハンドシェイク信号を用いてインターフェースされる．設計されたファンクション・ブロックは，これらの信号を用いて外部回路とインタフェースすることができる．これらの信号は，AccelChipのツールにより自動的に付加される．

入力データは自動生成された，ND_indata_in(New Data Flag)とRFND_indata_in(Ready For New Data)信号でハンドシェイクを行う．図15に示したように，ND_indata_in，RFND_indata_inの両信号がアクティブなときに，デザイン・ファンクション内にデータが取り込まれる．

出力データは，図16で示したようにDone_outdata_outと，DA_outdata_out(Data Accepted)信号でハンドシェイクされる．ファンクション回路部は，出力データが準備できたら，Done_outdataをアクティブにし外部回路に出力する．外部回路は，データを取り込んだらDA_outputをアクティブにし，データを受け取ったことを通知する．



AccelWare　ライブラリ

AccelWareは，論理合成可能なライブラリとして，Communication Tool Box，Signal Processing Tool Boxに対応した関数が提供されている．現在，表1の関数がサポートされ

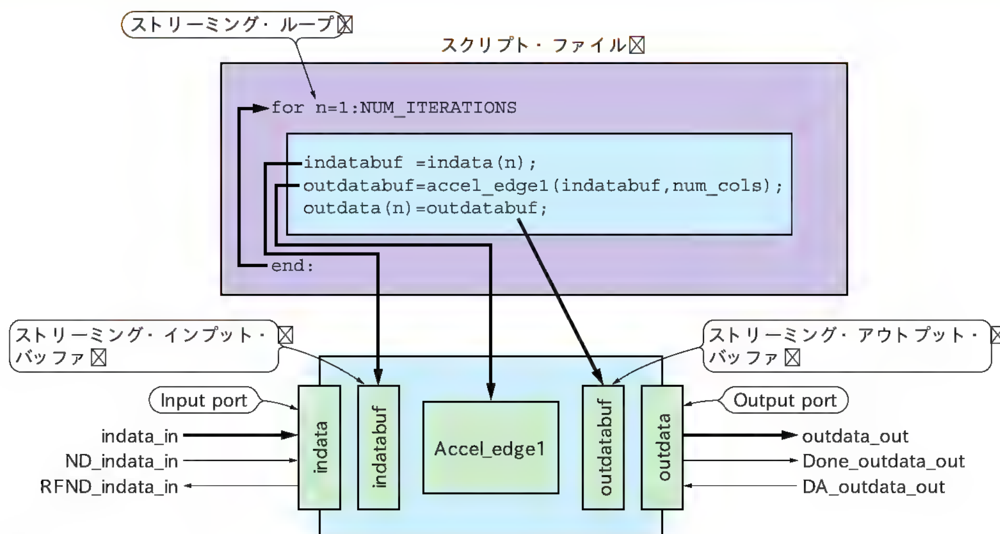


図14　ハードウェア上での対応

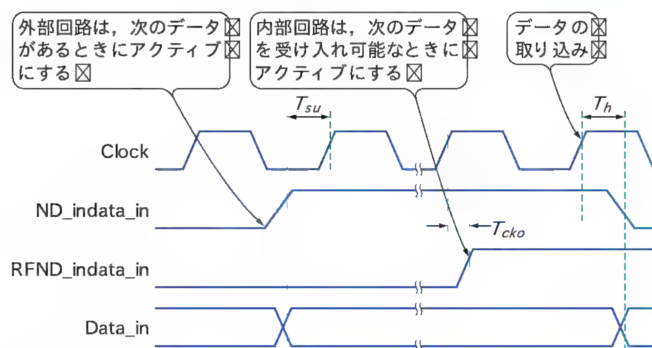


図15　入力ハンドシェイク

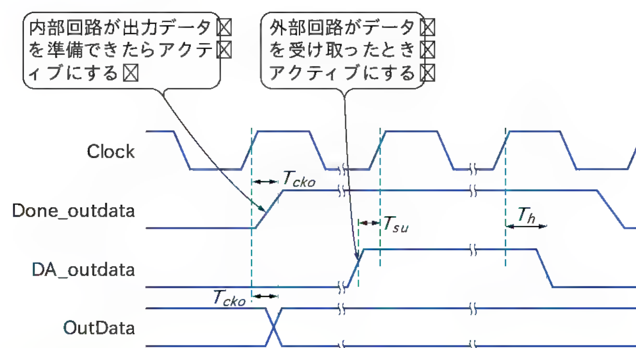


図16　出力ハンドシェイク

表1
AccelWareライブラリ

基本演算子	信号処理ブロック	通信ブロック
Absolute value	General-Purpose FIR Filter	Convolutional Deinterleaver
Angle(of complex number)	FFT	Convolutional Encoder
ArcCosine	IFFT	Convolutional Interleaver
ArcSine	Cascaded Integrator-Comb Decimator	Reed-Solomon BCH Encoder
ArcTangent	Cascaded Integrator-Comb Interpolator	A/D Sinc Compensation Filter
Cosine	Half-band FIR Filter	Raised Cosine Filter
Sine		
Tangent		
Square root		

ており，順次サポートされる関数は増えていく予定である．

これらの関数は，パラメータなどを設定変更することで，そのまま論理合成可能なMファイルが生成される．生成されたデザインは，デザイン・ファンクションそのものとして使用し，FPGA化することもできるし，ほかのデザイン・ファンクション内でデザインの一部として使用することもできる．また，ターゲットのチップに依存していないため，各ベンダのFPGA，ASIC，ストラクチャードASICに対して使用することができる．

まとめ

AccelChipは，Mファイルで設計したディジタル信号処理アルゴリズムを，デザイン・フロー・バーに示されるアイコンをクリックしていくだけで，RTLモデルの生成を自動的に行うことができ，従来，手作業で時間がかかっていたFPGA設計プロセスの時間短縮が行える．MATLABをで生成したアルゴリズムをFPGA上で実現したいと考えている技術者にとって，強力なツールである．

もちだ・まさゆき　(株)ロッキー



EDAツール＆回路設計自動化カンファレンス

DAC2004(第41回)San Diego展示見学レポート

倉重 克己

● 今年の日本の電子産業は期待できる!!

今年のDAC(Design Automation Conference)は，6月7日～11日に米国西海岸メキシコとの国境近くの港町San Diegoで開催された．筆者は，7日～9日の3日間，取引先との会議や友人との旧交を温め，その合間に展示だけを見て回った．DACは展示だけでなく同時に学会やセミナも大規模に行われているが，残念ながら参加する余裕はなかった．

以下に，印象に残ったこと，感じたことを率直に報告する．どこまで客観性があるかは大いに疑わしいが，訪問されなかった方々に雰囲気だけでもと考え，報告する．

まず，去年のDAC2003(Anaheim)より出展者，来訪者ともにかなり多いという感じだった．出展は案内書によると225社で実際去年より30社程度増え，来場者も15%程度増えたようである．日本からの訪問者も，去年は不景気のうえにSARS騒ぎまであって非常に少なかったが，今年はものすごく増えており，日本の電子産業のこの一年は期待できるのでは，とたいへんに喜ばしい状態だった．

● SystemCやSystemVerilogの趨勢やいかに

高位設計がどうなるかはだれもが興味があるところだと思うが，「SystemCは急速に普及するのか」，「SystemVerilogはどうなるのか」との疑問をもって見て回った．SystemCについてはForte DesignSystems，Celoxcaなどが合成コンパイラを展示しており，SystemCもモデリングだけでなく実装に大きく進んだ感じである．SystemVerilogは実質この1年の間に多くの会社がメイン・ストリームと判断したのか，いろいろなところでサポート済みと表明していた．これは多少は予測していたものの，実際にはそれ以上だった．シミュレーション，検証，合成とツール類はみな揃い，準備完了になっているのではと思う．いくつかの分派があるようだが，Synopsys主導で落ち着くとの意見が友人間では大半だった．

● ロジック設計で検証工程が8割を占める

検証系の出展が非常に多かった．検証が設計工程の8割といわれる時代にあって，シミュレーションだけに頼れないので当然と思う．出展者数の3割との感じだった．多くは，数学的な方法であるフォーマル検証，シンボリック・シミュレーション，ステート・スペース・サーチなど(筆者にとって)あまりなじみのない手段により実現されているツールである．ASIC設計の環境では普通に使用されてきていると思うが，安くなってきているので大規模FPGAの世界でも利用が進むのではと感じる．



今回のDACはIC設計に偏り，組み込みシステム設計やボード設計に関わるツールの出展が少なかった．DACに毎年来る組み込み業界やボード・メーカの友人の何人かからは，我々がこの展示会に来る意味がなくなってきたとの意見もあった．この分野の既存の設計手法の製品は行き渡り，シェアも固まり，出展しても意味がないのかもしれない．しかし，フォーマル検証ツールなどがFPGAやシステム設計に使われるようになり，SystemCをはじめとするビヘイビア水準の設計が盛り上がりの時期を迎えると事情が変わるかもしれない．

微細化のためとシステム設計化のための両方向で，アナログ系，実装系も元気がよく，出展も増えたようである．微細化のため，実装時の寄生*C*/*L*を抽出するのに，2Dでは不十分で完全な3Dの電磁界解析が必要になってきたということに少しおどろいたが，タイムリに起業するベンチャや素早い製品化のダイナミズムに感心する．

● 日本企業も先を読んだ出展に期待したい

日本または日系企業の出展は相変わらずきわめて少なく，存在感がなく少し将来に不安を感じる．総数4社(Applied Simulation Technologiy，CATS，Fujitsu，ZukenUSA)で，今年は1社減って1社増えたので増減0ではと思う．その1社はCATS社で，これを機会に日本を出て大いに発展していただきたい．



シリコン・バレーに本社はあっても，開発拠点，資金集めを含め実質は台湾という成功した会社が何社もあることを知っているが，友人によるとそういう会社は中国にもインドにもあるという．韓国もKAIST出身者を中心に大いに発展しそうである．出展ブースを見ても，インド系，台湾系/中国系が人員構成では半分を占めるのではないかと思うほど多い．日本は，産業界，学会をあげてまじめに，電子産業のマザーツールであるこの分野の育成を考える必要があるのではと感じた．

例年のことであるが，日本からの多くの訪問者はあまり小さなブースを訪問せず，大きなベンダのDACスイート通いが主に思えた．欧米の競合他社の導入評価がある程度定まってからの導入が主流のためだと思うが，自分の目でスタート・アップの企業の中から先端を探し，多少のリスクを取って自社に適用することも，優位な設計やコスト競争上，必要なのではないかと思う．

くらしげ・かつみ (有)インターリンク

「VxWorks」を使ったRTOS技術の基礎と応用



第8回

組み込みのためのC言語講座

高山 剛

C言語は非常に移植性に優れた言語です．私は以前，OSなしで構築された比較的大規模な組み込みシステムをVxWorksに移植したことがあります．コンパイラが「char型をデフォルトでsignedとして扱うかunsignedとして扱うか」の違いに依存して動かなかったコードが1か所ありましたが，それ以外は見事に移植できた経験があります．

ソフトウェアの再利用性は，たとえOS間に多少違いがあっても，C言語で記述すれば非常に高いといえます．

UNIX向けには，多種多様なC言語の教科書がたくさんあります．しかし，組み込み向けに考慮されたC言語の入門書はないように見受けます．経験あるエンジニアは自然に理解されているでしょうが，組み込み特有のC言語の使い方，移植性，最適化方法を列挙してみると，意外に多いことがわかりました．VxWorksだけでなく各種のRTOS向け，またはOSなしでソフトウェアを開発されているエンジニアの方すべてに役立てばと思います．

C言語とRTOSはベストマッチ

RTOS（Real Time OS）ではC言語がおもに使用されます．考えられる理由をあげると，次のことがあります．

▶アセンブラにもっとも近い言語である
▶ポインタを使ってI/Oに直接アクセスできる
▶コードが小さい
▶UNIXスタンダードで，学習しやすい
▶RTOSをCの関数で実現できる

家電製品などでは，対コストということでアセンブラも幅広く使用されています．しかし，32ビットCPUを使用し，数百Kバイトを超えるようなアプリケーションになると，C言語のほうがメンテナンス性，他プロセッサへの移植性，生産性といった面で圧倒的に優れています．

パフォーマンスにおいても，アセンブラがC言語より性能が出るのは当たり前です．しかし，C言語から生成されるコードは，コーリング・シーケンスと呼ばれる取り決めでコードが生成されるため，このコーリング・シーケンスに従い，C言語からアセンブラを呼び出したり，逆にアセンブラから複雑な条件分岐や複雑な計算だけ（呼び出し頻度が稀な場合に）C言語を呼び出すなど，性能の関係する箇所だけに部分的にアセンブラを使用すれば，システム全体の性能はアセンブラだけで記述するのとC言語＋アセンブラをミックスした場合も大差がないといえます．

とくに，FAやインダストリアル分野では，顧客別にカスタマイズをする必要があったり，顧客の都合によって仕様が変更される場合が多々あり，メンテナンス性を考えればまちがいなくC/C++が使われるでしょう．

C++はBetter C（C言語として使用するが，C++の良いところをプログラマが選択して使用する）として，また本格的なオブジェクト指向言語として使用されています．しかし，パフォーマンス，サイズ，シンプルさと学習のしやすさでC言語で構築されることが非常に多いのが現実です．

さてRTOSとC言語の関係ですが，C言語では，言語的に足りないところは関数で実現します．RTOSのシステム・コールも関数で，カーネルのスケジューラもCの関数で実現されています．もちろん，タスク・コンテキスト・スイッチでのレジスタのセーブ・リストアなど，C言語で記述不可能なところはアセンブラで実現されています．

C言語でタスクの切り替えをどうするのだろうと思われるかもしれませんが，UNIXのsetjump，longjmpといった関数はCPUのレジスタをそっくりセーブ・リストアして，プログラムの制御を，関数の枠を越えてジャンプできますが，RTOSのカーネルのタスク・スイッチの実装もこれに似ています．

UNIXアプリケーションと組み込みでのCプログラミングの違い

UNIXアプリケーションとRTOSでのCプログラミングの違いは何でしょうか．一つにプロセス・モデルとスレッド・モデルの違いがあります（図1）．プロセス・モデルは，論理アドレス空間を必要とするためMMUを必要とし，プロセス間，カーネル・プロセス間でのスイッチ，メッセージ交換でオーバヘッドが大きくなります．

RTOSは，そのオーバヘッドが無視できない組み込み用途向

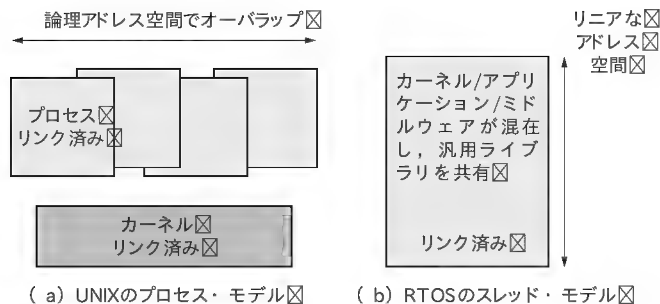


図1　UNIXのプロセス・モデルとRTOSのスレッド・モデルの違い

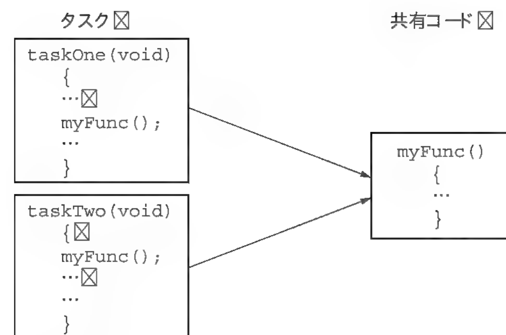


図2　共有コード

けにスレッド・モデルで構築されています．スレッド・モデルではポインタの交換でカーネル，割り込み，タスク間で効率よくデータを交換できます．

RTOSは，高パフォーマンス，スモール・コード・サイズの点で有利なスレッド・モデルであるため，図2のように一つのコードが，複数のタスクから呼び出されることがあります．たとえば，多くのタスクが`printf()`を呼び出しますが，システム中には`printf()`は一つしかありません．このように複数のタスク・コンテキストから呼び出される関数は，作業用のグローバル変数やスタティック変数が衝突しないようにコーディングされていなければなりません．これをリエントラント（再入可能）と呼びます．プロセス・モデルであれば，これらの関数は，プロセスごとに一つもっていますが，スレッド・モデルではシステム中の一つを共有してもちます（コード・サイズが小さくなる理由がここにある）．

さて，リエントラント（再入可能）にコーディングするためにはどうすれば良いかというと，次のようにコーディングすることが求められます．

▶ スタック上にデータ領域が確保されるオート変数を使う
▶ セマフォで，グローバル変数，スタティック変数を保護する
▶ タスク変数（VxWorks独自の機能の一つ，タスク・スイッチのたびに退避・復旧を行い，グローバル変数を共有していてもあたかも複数のインスタンスがあるようにふるまうことができる）を使う

要は，オート変数を使い，グローバル変数はなるべく使わず，大きな配列は`malloc()`で確保し，ほかのOSのアプリケーションや，レガシなコード，コード・ジェネレータが生成したコードをRTOSにもってきて修正せずに使いたい場合など，仕方ない場合に，タスク変数を使うことになります．タスク変数はオーバヘッドが大きいのでなるべく使わないようにしてください．

システムに原因不明の不具合が起こった場合，コードがリエントラントでないことが原因で発生した可能性があります．チェック項目として，ぜひ記憶しておいてください．

二つ目に，スタックについてつねに意識が必要です．UNIXでは，デマンド・ページングなのでスタック・サイズを気にする必要はありませんが，リアルタイム性が重要なRTOSではデマンド・ページングを採用できません．リアルタイム応答性が必要な重要なサブルーチンを呼び出す際に，たまたまスタックに実メモリがなくデマンド・ページングが発生した場合，OSに大きなオーバヘッドを発生させます．

RTOSでは，スタックはタスク生成時に固定のサイズを与え，プログラムは，オート変数で大きな配列は確保しないように心がける必要があります．大きな配列が必要ならば，`malloc()`を呼び出す必要があるでしょう．

最後に，RTOSでは直接ハードウェアにアクセスを行います．UNIXアプリケーションはハードウェアへの直接アクセスは許されません．RTOSでは，ブロック・デバイス，キャラクタ・デバイス，Ethernetのデバイスに対してはドライバを経由してアクセスすることで，ファイル・システムやI/Oシステムのサービスの恩恵を受けるので，ハードウェアへのアクセスを行うことはないのですが，スイッチング・モータ，アクチュエータ，センサ，A-D/D-Aコンバータなどのデバイスに関してはデバイス・ドライバを介さず，直接アクセスすることが多いのです．これは，RTOSではアプリケーション自体が一種のドライバのような性格であるからです．

I/Oへのアクセスについては，次の項で詳しく述べます．

I/Oへのアクセス

コンピュータ（CPU）は，データを入出力するI/Oがなければ何も役に立ちません．CPUはI/Oを操作するためのレジスタをCPUのメモリ空間に割り当て（メモリ・マップドI/O）入出力を行います（x86アーキテクチャ，また一部のアーキテクチャでは，別空間をもつプロセッサもある）．

では，シリアルのデバイスのアドレスが`0xf0001000`番地に存在する場合，どのようにC言語で記述すればよいでしょうか．

じつは，私が社会人になりたてのころ，学習用にちょっとしたゲームをC言語でUNIXマシン用に作ったことはあったのですが，フロッピ・ディスク・ドライブのドライバを書くことになって，さてどうしたらI/Oにアクセスできるのやらと途方に暮れたことがあります．先輩に次のように教えてもらって，そんなことができるのかと思った記憶が残っています．

```
unsigned int *pointer;
pointer = (unsigned int * ) 0xf0001000;
*pointer = BIT0;
*pointer |= BIT2;
```

ポインタというのは，配列を効率よくアクセスするためのものと理解していたので，絶対アドレスを代入するとは想像もつかなかったのです．

しかし，上の例ではCコンパイラはpointerがI/Oだとは解釈していません．最近のコンパイラは最適化技術が進んでいるので最新コンパイラを使うと，意図したビット0をセットし，その後，ビット2をセットするのではなく，コンパイラは最適化して，いきなり，ビット0，ビット2を同時セットするコードを生成するかもしれません．これではI/Oを期待どおりに制御できません．

ANSI Cでは，最適化が行われてもよい変数と，最適化が行われては困る変数を指定できるようにvolatile宣言が追加されています．したがって，

```
volatile unsigned int *pointer;
pointer = (unsigned int * ) 0xf0001000;
*pointer = BIT0;
*pointer |= BIT2;
```

とすべきです(コンパイラに-fvolatileというオプションがあり，そのモジュール全体をvolatileにしてしまう方法もある)．また，性能的に支障がなければ，次のように関数を用意して関数経由でI/Oにアクセスすれば，

```
sysIoOut(adr, data)
```

とI/Oアクセスする箇所でブレーク・ポイントの設定が容易になり便利です．

C言語のポータビリティ

unsigned intは32ビットCPUではたいていの場合32ビットですが，ほかのCPUアーキテクチャで16ビットだったらどうでしょうか．C言語のANSI規格ではint型のサイズは処理系依存，CPU依存となっているため，いつも32ビットとは限りません．

VxWorksでは，

```
typedef unsigned int UINT32;
```

のようにINT8，INT16，INT32，UINT8，UNIT16，UINT32などをVxWorksのヘッダで定義していて，4バイトのI/OだったらUINT32，1バイトのI/OだったらUINT8を使うようにサイズを明示するような取り決めを行っています．これで確実に互換性を保てるようにしています．

```
volatile UINT32 *pointer;
pointer = (UINT32 * ) 0xf0001000;
*pointer = BIT0;
*pointer |= BIT2;
```

が互換性に優れたコードといえます．

また，あるドライバでは，I/Oレジスタのマッピングを次のように，構造体で表現して構造体ポインタでアクセスするドライバも存在します．

```
struct iomap {
 UINT32 int1;
 INT16  aShort;
 INT8   aByte;
}
```

I/Oのレジスタの点数が多くなると，確かにコーディングがすっきりします．また，ポートの数が多いデバイスでは，同じレジスタがポートの数だけ存在します．構造体によりコーディングしやすくなる場合があります．

この場合，コンパイラがプログラマの意図に反して余計なパディングを詰め込まないように，アラインの設定が必要になります．ただ，このアラインの指定はコンパイラにより差異があるため，注意が必要です．

VxWorksの場合は，複数のコンパイラ(GCC，Diab，その他)に対応するため，マクロを用意しています．

```
struct iomap {
 UINT32 int1;
 INT16  aShort;
 INT8   aByte;
} _WRS_PACK_ALIGN(1);
```

経験の深いエンジニアの方は，さらに一つのレジスタのビットさえも，C言語のBitfieldを使いこなしてI/Oの制御ができると思われるでしょう．しかし，コンパイラによってはビットの割り付けをMSBから始めたり，LSBから始める場合があります．このため，VxWorksではBitfieldの使用を勧めていません．次のようにビットを示すマクロを定義することを推奨しています．

```
#define  ABYTE_ERROR  0x01
#define  ABYTE_OFLOW  0x02
#define  ABYTE_UFLOW  0x04
#define  ABYTE_DMA    0x08
#define  ABYTE_POLL   0x10
```

最後に，ほかに注意すべき点をあげておきます．これで，さらに移植性の高いコードになるでしょう．

● if(fp = fopen("abcd", "r") == NULL)

まちがって，if(fp = fopen("abcd", "r") <=NULL)としないよう注意してください．あるCPUアーキテクチャではアドレス空間のMSBが1の空間にメモリがある場合があるので，ファイルのオープンに成功したのに負の値が返り，このコーディングではファイルのオープンに成功しても，いつもエラーとして処理されてしまいます．

● Bi-endian code

CPUにはビッグ・エンディアンとリトル・エンディアンの2

Column 1

コーディング規約

VxWorksのよいところの一つに，厳密なコーディング規約がVxWorksの全コードに適用されており，VxWorksの関数の詳細なマニュアルは，ソース・コードに埋め込まれて，mangenというツールでソース・コードから自動生成されるという点があります．

ソース・コードとドキュメントが一体になっており，仕様書，ドキュメント，ソース・コードが一元的に管理されるのでたいへんメンテナンス性がよいといえます(リストA)．このmangenはVxWorksの製品の中のtarget/unsupported/tools/mangenで公開されているので，ぜひ自分自身のアプリケーション・コードもコーディング規約に従って，mangenを使って関数の仕様書を作成するとよいでしょう．

特に関数の引き数を一行に一つ書いて，右にコメントを書いていくスタイルは便利だと思います．mangenはこのスタイルで書いておくと次のように自動的に引き数の説明文にしてくれます(リストB)．

リストA　ソース・コードの一例

```
/**********************************************
*
* memcmp - compare two blocks of memory (ANSI)
*
* This routine compares successive elements
                    from two arrays of `unsigned char',
* beginning at the addresses <s1> and <s2>
                      (both of size <n>), until it finds
* elements that are not equal.
*
* INCLUDE FILES: string.h
*
* RETURNS:
* If all elements are equal, zero.
                   If elements differ and the differing
* element from <s1> is greater than
                     the element from <s2>, the routine
* returns a positive number; otherwise,
                          it returns a negative number.
*/

int memcmp
    (
    const void * s1,            /* array 1 */
    const void * s2,            /* array 2 */
    size_t     n
              /* size of memory to compare */
    )
    {
```

リストB　mangenで自動生成された.manページの例

```
memcmp(?)
NAME
memcmp(?) - compare two blocks
                                       of memory (ANSI)
SYNOPSIS
int memcmp
    (
    const void * s1,          /* array 1 */
    const void * s2,          /* array 2 */
    size_t       n
            /* size of memory to compare */
    )
DESCRIPTION
This routine compares successive elements from
          two arrays of unsigned char,
          beginning at the addresses s1 and s2
          (both of size n),. <<以降略>>
```

種類が存在することはよく知られています．では，エンディアンに依存しないコードとはどのように記述すべきでしょうか．ネットワークのコードでは，ビッグ・エンディアンがスタンダードになっており，

```
ntohs()
htons()
ntohl()
htonl()
```

のマクロを使う必要があります．

リトル・エンディアンのntohl()の定義は次のとおりです．

```
#define ntohl(x) ¥
            ((((x) & 0x000000ff) << 24) | ¥
             (((x) & 0x0000ff00) <<  8) | ¥
             (((x) & 0x00ff0000) >>  8) | ¥
             (((x) & 0xff000000) >> 24))
```

ファイルのデータがバイナリ・ファイルであれば，このマクロを使いエンディアンを統一するか，もしくは文字列を使うべきでしょう．

リトル・エンディアンを使っているエンジニアは，これらをご存じでしょう．ビッグ・エンディアンのエンジニアは，ネットワーク・コードがエンディアンを考慮していなくても動いてしまうので意識しないかもしれませんが，リトル・エンディアンのプロセッサへの移植の際に問題になります．ネットワークでは，ポート番号，それからデータの中のバイナリ・データが影響を受けます．

● alloca()の使用を避ける

あるコンパイラはalloca()関数を，C言語の拡張機能としてサポートしており，スタックから可変長のメモリを確保します．RTOSではスタックから大きなメモリ・ブロックを確保することは，RTOSがUNIXのように動的にスタックを拡張することができないため潜在的に危険が増すことになります．したがってRTOSではalloca()を使用しないほうがよいでしょう．

● コーディング規約を決める

VxWorksのようにスレッド・モデルのOSは，アプリケーションが大きくなるとC言語のシンボルの重複が問題になります．そこで，コーディング規約を決めておいて変数名，関数名，マクロ名のネーミングの仕方を統一することでシンボル名の重複の問題を解決します．これがひいては，再利用性，移植性を高くします．

ここでは，VxWorksで取り決められているコーディング規

約を紹介しましょう．ほかのOSを使用しても役立ちそうです．

① 関数，グローバル変数には，モジュール名単位に3文字でそのモジュールを示す文字列を決めて接頭辞とする

② 関数は，モジュール名を示す文字と，名詞，動詞と続ける．さらに単語の頭文字だけ大文字を使う

```
modObjFind()
```

③ マクロ名は，大文字で統一する．単語間は_（アンダスコア）を入れる

```
MOD_MAX_COUNT
```

④ モジュール内でしか使わない変数，ローカル変数はstatic宣言する

```
static subFooFind();
```

⑤ ポインタはほかの変数と区別できるように，接頭辞としてpを付ける．ポインタのポインタはpp

```
FOO_NODE *    pFooNode;
FOO_NODE **   ppFooNode;
```

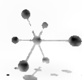
⑥ 最後に，ヘッダ・ファイルはアプリケーションが巨大になると，予期せず入れ子になってしまう場合がある．構造化されていないと二重定義がたびたび発生して，ひいては移植性，再利用性が低くなってしまう．しかし，次のようにヘッダ・ファイル内を#ifndefで囲むことによってインクルード文の重複を防ぐことができる

```
#ifndef __INCfooLibh
#define __INCfooLibh
<fooLib.h の中身>
#endif /* __INCfooLibh */
```

● デバッガによるCとアセンブラのミックス表示

組み込みでのデバッグでは，C言語のソース・コードをシングル・ステップでデバッグをしていて，「コーディングは確かに正しいはずなのに期待どおりシステムが動作しない」場面が時折あります．

こういうときはデバッガ（もしくはCコンパイラのオプションで）をC言語，アセンブラのミックス表示にしてデバッグしてみましょう．ソース・コードであるC言語と生成されたアセンブラ・コードを対比してデバッグすることでI/Oに意図した手順でアクセスしていないなどの問題点を見つけられるかもしれません．C言語が，どのようにアセンブラに変換されるか知っておくのも勉強になります．



パフォーマンスの最適化

製品の売れ行きを左右する差別化要因は，価格や機能といった場合が多いようですが，ある種の組み込み機器では性能だけが差別化要因というものもあります．

性能改善にはハードウェアによるアクセラレータという手段もありますが，ここでは，組み込み特有のソフトウェア上の改善点を紹介します．

● 見かけ上の高速化

見かけ上の高速化の一例を紹介します．かつて昔，画面のスクロールの速度がPCアプリケーションの差別化になっていたころ，見えないくらい速いスクロールが話題を呼んだことがあります．しかし，実はまじめにスクロールさせていたのではなく，適当にはしょってスクロールしているかのように見せかけていたという話を聞いたことがあります．これも一つの最適化だと思います．

組み込みのためのC言語講座がテーマなのに，発想の転換が重要だと言うのは不適切かもしれませんが，組み込み機器では，パソコンと違って，それぞれ組み込み機器固有の特徴を利用できるからです．

たとえばディジタル・カメラでは，電源ONした後，起動画面だけを映し出し，カメラのファインダをのぞき込むまでの間に周辺機器をイニシャライズし，被写体を捉える間にファイル・システムをマウント状態にしておけば，起動時間を見かけ上，高速にできるでしょう．

VxWorksやほかのRTOSではUNIXと違って，ファイル・システムをアプリケーションで初期化できます．

VxWorksの場合，ネットワークはusrNetInitを呼び出すことで後から初期化できます．UNIXと違って，見かけ上のブートの高速化が簡単にできるところがRTOSらしいところです．

● ボトルネックを知る

まずは，システムのボトルネックを知る必要があります．コードの量が多い，ループ回数が多いなど，なんとなく遅いであろうと勘に頼る手法は結局は非効率です．性能が達成できなかった場合，最後には実測するはめになります．

実測で個々のパートの実行時間を計測して初めてシステムのボトルネックがわかります．たった1行のコードなのに浮動小数点演算が異常に遅かったり，ハードウェア（メモリ，バス，ペリフェラル）で思わぬ遅延があったなど，机上の計算と異なる事例は多々あります．

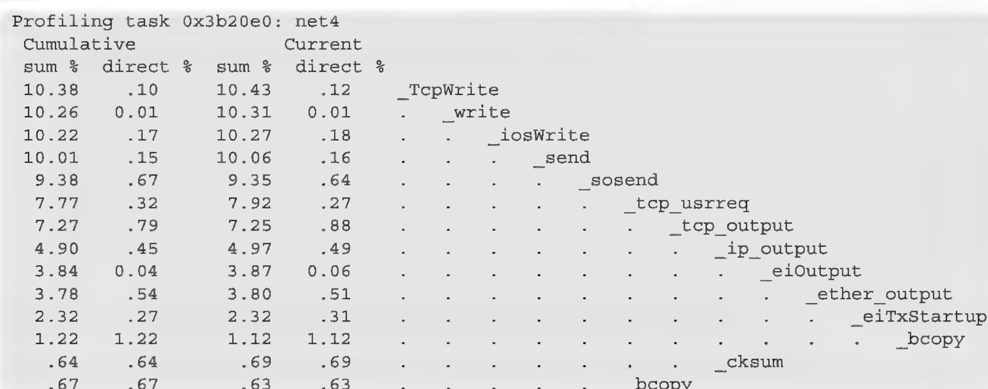
ロジック・アナライザでは，生き物のようなソフトウェアの性能をあらゆる条件下で計測するには非効率ですが，VxWorksでは，spyコマンドでタスク，割り込みのCPU使用率，timexコマンドで特定関数の実行時間を計測できます．図3のProfileScope（RTI社）を使えば，CPU使用率を関数単位で詳細に知ることもできます．

● キャッシュの用法

最近のプロセッサは周波数が高くなって，メモリのアクセス速度より格段に速く動作します．言い換えれば，メモリが相対的に遅くなったといえます．それゆえ，命令/データ・キャッシュへのヒット率がパフォーマンスに大きく影響することになります．

ソフトウェア的に高効率（高キャッシュ・ヒット率）なキャッシュの用法を以下にあげました．

▶ プログラムを小さくすることで，命令キャッシュへのヒット率向上



```
Profiling task 0x3b20e0: net4
 Cumulative             Current
 sum %   direct %   sum %   direct %
 10.38     .10      10.43     .12     _TcpWrite
 10.26    0.01      10.31    0.01     .   _write
 10.22     .17      10.27     .18     .   .   _iosWrite
 10.01     .15      10.06     .16     .   .   .   _send
  9.38     .67       9.35     .64     .   .   .   .   _sosend
  7.77     .32       7.92     .27     .   .   .   .   .   _tcp_usrreq
  7.27     .79       7.25     .88     .   .   .   .   .   .   _tcp_output
  4.90     .45       4.97     .49     .   .   .   .   .   .   .   _ip_output
  3.84    0.04       3.87    0.06     .   .   .   .   .   .   .   .   _eiOutput
  3.78     .54       3.80     .51     .   .   .   .   .   .   .   .   .   _ether_output
  2.32     .27       2.32     .31     .   .   .   .   .   .   .   .   .   .   _eiTxStartup
  1.22    1.22       1.12    1.12     .   .   .   .   .   .   .   .   .   .   .   _bcopy
   .64     .64        .69     .69     .   .   .   .   .   .   .   _cksum
   .67     .67        .63     .63     .   .   .   .   .   _bcopy
```

図3
ProfileScopeの表示例

C言語，アルゴリズム・レベルでのコード・サイズの縮小は，ほかの書籍に譲ることにして，今回は後ほど取り上げるアセンブラ化によるコード・サイズの縮小の効果に注目したいと思います．

▶ キャッシュと実メモリは最適化されたデータ転送がなされているか

キャッシュがミス・ヒットした場合，実メモリからデータ・コピーが起こりますが，フォン・ノイマン型のコンピュータのプログラムの局所性を利用して，たいていのキャッシュ・システムはRAM，ROMからバースト転送，ページ読み込みを行います．メモリ・コントローラが最適に設定されているかが性能に影響します．

C言語とは直接関係はありませんが，ハードウェアの知識として必要でしょう．

▶ キャッシュ・スラッシングは起こっていないか

キャッシュとは，実メモリのアドレスとデータの関連付けをCPU内部にもつことで高速にアクセスできる機構です．キャッシュ・スラッシングとは，このアドレスとメモリの関連付けのしくみの巧拙によって引き起こされる問題で，最悪の場合，キャッシュを使わない場合より遅くなってしまう場合があります．

具体的に説明すると，CPUのキャッシュは，CPUを高速に動作させることが目的ですから，できるだけ簡単なしくみで実装しないと意味がありません．いちばん単純なダイレクト・マップ・キャッシュと呼ばれるキャッシュでは，たとえば，(説明しやすくするため架空のキャッシュを想定)アドレス(32ビットとして)として，アドレス8～11ビットをインデックスとしたキャッシュとします．

この場合，CPU(キャッシュを含む)は，図4のテーブルをもつことになります．

インデックスは8～11ビットを使い，4ビットなので16エントリとなります．

ダイレクト・マップ・キャッシュでは，アドレス`0x12345678`にアクセスが必要だった場合，ビット8～11をインデックスとして(この場合は`0110`)，すなわち6がインデックスになります．インデックス6のタグ部にあるタグが，アドレスの31～12ビット，この例では`12345`と一致していれば，キャッシュ・ヒットと見なし，実メモリではなくキャッシュのデータ部よりCPUに読み込みます．

このダイレクト・マップ・キャッシュの場合，問題なのは次のようなプログラムです．

```
Task1
UINT8 dt1[256],
for (i=0;i<256:i++)
```

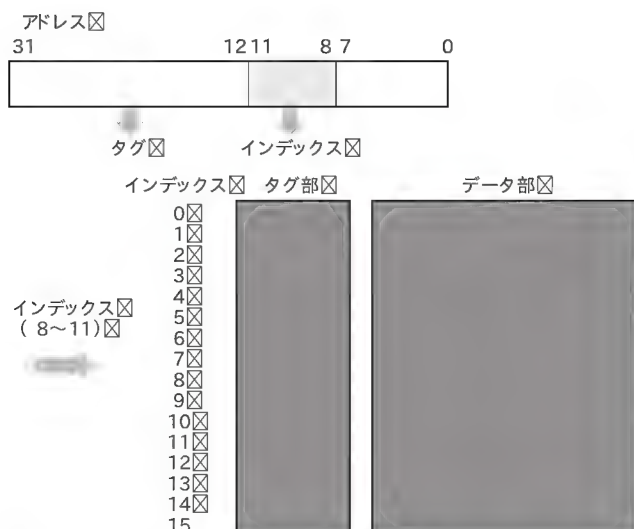


図4　インデックス・テーブル

```
   sum+ = dt1[i];

  Task2
  UINT8 dt2[256];
  for (i=0;i<256:i++)
   sum+ = dt2[i];
```

かりにdt1，dt2のアドレスが，それぞれ0x12345600と0x12355600でTask1，Task2が交互に動作する場合はどうなるでしょうか．

どちらもインデックスが6のため，Task1とTask2のタスクが切り替わるたびに，dt1，dt2へのアクセスでキャッシュ・ミス・ヒットとなってしまいます．これが頻繁に起こる場合，性能が極端に低下します．これがキャッシュ・スラッシングと呼ばれる現象です．

RTOSの場合，TCB（タスク・コントロール・ブロック）でキャッシュ・スラッシングが起こる場合があるので，注意が必要です．これを避ける方法は，TCBを重ならないように配置することも考えられますが，ほかにもスタックやアプリケーション・データでも起こりえます．さらにプロセッサ，キャッシュ・アーキテクチャごとにブロック単位が千差万別です．ほかにも複数タスクが存在するのでOSやアプリケーションで厳密に避けることは困難です．この問題を回避するには，セット・アソシエイティブ・キャッシュ，フルセット・アソシエイティブ・キャッシュを使うほかになさそうです．

まったく同じプログラムなのに，速い場合，遅い場合があるとき，このキャッシュ・スラッシングを疑ってください．

● アセンブラ化で高速化する

Cコンパイラの性能が高くなって，アセンブラ化の必要性は減ってきましたが，限界まで性能を出したい場合はアセンブラ化の検討が必要でしょう．

「なぜ，アセンブラ化すると高速化できるか？」から整理すると，

① 変数が取り得る値の範囲をプログラマは知り得ているので，最適のインストラクションを選べる．不要な符合拡張を省ける
② 使用頻度の高い変数をレジスタに割り当て効率良く使用できるので，スタックへ割り当てる変数を減らせる
③ コンパイラが使わないような特殊なインストラクション（たとえばマルチメディア系命令）が使える

①はプログラムのコード・サイズが小さくなり，キャッシュ・ヒット率が高くなるというメリットがあります．②は不要なメモリ・アクセス（メモリ・アクセスのオーバヘッドがいちばん大きい）が減り，性能の改善が期待できます．③によってもサイズが結果的に小さくなり，キャッシュ・ヒット率が向上するでしょう．

● Cとアセンブラのインターフェースは？

前述したように，全部のコードをアセンブラで書く必要性はまったくありません．99％をCで書いて1％をアセンブラで書いても，性能はアセンブラですべて書いた場合と限りなく近く構築することができます．そのためには，C言語からアセンブラ，アセンブラからC言語を呼び出す必要があります．

Cコンパイラは，関数間のコーリング・シーケンスを厳密に規定しています．アセンブラのコードを，このコーリング・シーケンスに基づいて記述することでCコンパイラのリンカは，C言語関数とアセンブラで記述された関数（コーリング・シーケンスに基づいて記述されれば立派なC言語の関数）をリンク可能になります．

以下にコーリング・シーケンスを理解する上で必要なキーワードを説明します．

● 引き数渡し

CPUによって引き数渡しに用いられるレジスタが決まっています．レジスタが多数あれば，あるレジスタ番号から，引き数1，2，3，…と割り当てられます．引き数渡しに用いられるレジスタの数にはCPUごとに依存します．それ以上の引き数はスタックを経由します．

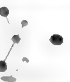
● 戻り値

関数の戻り値もレジスタが決まっています．通常二つのレジスタが割り当てられ，char型，short型，int型は一つのレジスタを使用し，double型の場合，二つのレジスタが使用されます．

● テンポラリ・レジスタ，パーマネント・レジスタ

RISCプロセッサの場合，多数のレジスタをもっていますが，それぞれ役割があり，効率良く使えるように組織化されています．

RISCプロセッサは，CPUの周波数を高くして，多数のレジスタを内部にもつことで，メモリ・アクセスの頻度を軽減し



オーバヘッドを減らそうというアーキテクチャです．そのため，アプリケーションが汎用に使えるレジスタをテンポラリ・レジスタとパーマネント・レジスタに分けて使用します．

パーマネント・レジスタとは，関数内で使用する場合，かならずセーブ，リストア(保存，回復)が必要になります．

テンポラリ・レジスタとは，関数内で破壊してしまってもよいレジスタを意味します．

そのため，Cコンパイラはテンポラリ・レジスタを優先して使い，場合によっては複数のオート変数を一つのレジスタに割り当てようと試みます．

パーマネント・レジスタは，テンポラリ・レジスタだけではレジスタが足りなくなった場合に使用されます．ただし，使用する場合，かならず使用するレジスタについてセーブ・リストアが必要です．というのは呼び出し側の関数はパーネント・レジスタについてはどんな関数を呼び出しても絶対に破壊されないという前提で，コンパイラがアセンブリ・コードを生成しているからです．

テンポラリ・レジスタ，パーマネント・レジスタと2種類に分けられていることでメモリ・アクセスの回数を減らすくふうがなされていることに注目してください．

話はそれますが，テンポラリ・レジスタとパーマネント・レジスタの区別は，OSのカーネル自体もメリットを享受しています．たとえば，割り込みが発生して，C言語を呼び出す場合，すべてのレジスタをセーブする必要はなく，テンポラリ・レジスタのみをセーブ・リストアします．パーマネント・レジスタをCの各関数が使用する場合，その関数自身が必要なだけのパーマネント・レジスタをセーブ・リストアするからです．ほかにもVxWorksのsemTake/Give，taskDelayなど，タスク・スイッチを引き起こす関数の場合，TCBにテンポラリ・レジスタをセーブしません．これはsemTake/Give，taskDelayなどを呼び出す関数は，テンポラリ・レジスタについて，これらの関数によって破壊されることを承知で呼び出して

Column 2

フェイル・セーフの処理

組み込みシステムでは，フェイル・セーフなシステムが要求されます．フェイル・セーフとは，人，機械，コンピュータ・ハードウェア，ソフトウェアがまちがえたり，故障した場合でも，問題を認識でき，最悪の障害を可能な限り防ぎ安全を確保することにあります．ここでは，アプリケーション・ソフトウェアの不具合でCPUが例外を発生した場合，どのようにハンドリングするかを紹介します．

VxWorksでは，例外処理はUNIX互換のsignalを使うことで処理できます．FA分野では，このようなコードをシステムに組み込み，ソフトウェアの問題だけでなく，ハードウェア(たとえば，過酷な環境ではハード的にメモリの内容が壊れることもあり得る)で問題が起こった場合，システムを一時停止し，問題が起こったことの通知，問題を事後解析するために情報の確保，リブートなどによる確実な復旧処理が必要になります．

リスト C　例外処理をシグナル・ハンドラで処理し，longjmpで大域ジャンプしてメイン・ループに戻るサンプル・コード

```
#include "vxWorks.h"
#include "signal.h"
#include "sigLib.h"
#include "taskLib.h"
#include "memLib.h"

#include "setjmp.h"
#include "regs.h"

extern sigHandler(int sig , int code ,
                struct sigcontext *);
jmp_buf save_env;

test ()
{
SIGVEC sigv;
SIGVEC psigv;
int er;

        /* signal handling setup */
        (FUNCPTR) sigv.sv_handler =
                        (FUNCPTR) sigHandler ;
        sigv.sv_mask = 0;
        sigv.sv_flags = 0;

        er = sigvec ( SIGSEGV  ,&sigv , &psigv );
        if ( er ) logMsg ( "sigvec return %d ",er );
        /* application main loop */
        for ( ;; )
            {
                taskDelay(200);
                if ( 0 == setjmp ( save_env ) )
                {
                do_work1();
                do_work2();
                } else {
                /* exception! */
                printf ( "Exception¥n" );
                }
        }
}

sigHandler ( int sig , int code ,
                            struct sigcontext *pSigCont )
        {

        logMsg ( "Problem PC %x¥n" ,
                            pSigCont->sc_pregs->pc );

        longjmp ( & save_env,1);

        }

do_work1 ()  { printf ("dowork1¥n"); }
do_work2 ()
    {
    int *p;
    int sum = 0;

    /* 問題を引き起こすコードの例 */
    for ( p= 0 ; ; p += 0x10000 )
        sum += *p;
    }
```

いるので，テンポラリ・レジスタの破壊に依存しないようにCコンパイラがコードを生成しているからです．

● アセンブラ用テンポラリ・レジスタ

RISCプロセッサでは，固定命令長というアーキテクチャのためCISCでは1命令で済む命令も複数の命令で記述しなければなりません．そこでアセンブラは複合命令という一種のマクロ命令をもっています．ここで，どうしても一つのレジスタを作業用に必要になる場合があります．これがアセンブラ用テンポラリ・レジスタです〔MIPSでは`$1(AT)`〕．

複合命令を理解して局所的に使う場合は使ってもかまわないのですが，関数内でローカル変数として割り当てるのは，どこで複合命令がこのレジスタを使用しているか，また将来だれがコードを変更するかわからないので使用するのは絶対にやめましょう．

● スタック・ポインタ，スタック変数

パーマネント・レジスタを使用する場合，スタックにセーブする必要があります．また，テンポラリ・レジスタ，パーマネント・レジスタでも変数が足りなくなった場合，配列やバッファをスタックに確保することができます．

スタックを使用する場合は，先にスタックを進めて，それから使用するのがルールとなっています．

● FPレジスタ

CPUのレジスタ・マニュアルを見るとFPというレジスタを見かけることがあります．これは，GDBなどのデバッガでスタックをトレースすることで，どこの関数から現在の関数まで呼び出されてきたかを過去にさかのぼってトレースする機能を実現するために存在します．

FPレジスタとは，スタック・フレーム（一つの関数が使うスタック・エリア）を示すアドレスを保持しています．スタック・フレーム内にもFPレジスタの内容を保持することで，スタック・フレームをリスト構造にすることができ，デバッガはスタック・フレームを簡単にトレースできます．

FPレジスタは，デバッグ目的で使用されるので，コンパイラのオプションで，このFPレジスタを使用しないようにすることも可能です．

アセンブラでコードを記述する場合も，FPレジスタを無視してもかまいません．ただ，アセンブラのコードからCの関数を呼び出し．そのC言語をデバッガでデバッグできるようにすることが重要と思われる場合は，FPレジスタをコーリング・シーケンスに従って使用するとよいでしょう．

● アセンブル生成オプション

Cコンパイラには，アセンブラ・コードを生成するオプションをもっています．GCC，Diabコンパイラでは`-S`オプションを指定することでアセンブラ・コードを出力します．C言語で使用する引き数や呼び出す関数だけを記述して，`-S`オプションでアセンブラ・コードを手に入れ，これをベースにアセンブラ・コードを作ると，簡単にアセンブラに挑戦できると思います．

参考文献
(1) Wind River, VxWorks BSP Developer's Guide, 5.5

たかやま・たけし　ウインドリバー（株）
takeshi.takayama@windriver.com

Interface BackNumber

2003年		2004年	
7月号	高速バスシステムの徹底研究	1月号	CD-ROM付き 基礎からわかるPCI&PCI-X活用技法
8月号	別冊付録付き 現代コンピュータ技術の基礎	2月号	別冊付録付き C++テンプレートプログラミングの世界
9月号	CD-ROM付き C/C++によるハードウェア設計入門	3月号	Cプログラミングの基礎知識
10月号	詳細マイクロプロセッサ―パイプラインとスーパースカラ	4月号	作りながら学ぶEthernet活用技法
11月号	マイクロプロセッサ技術の基本	5月号	別冊付録付き 組み込みシステムの世界へようこそ！
12月号	別冊付録付き 具体例で学ぶ組み込みソフトの再利用技術	6月号	ようこそ二足歩行ロボット制御の世界へ
		7月号	MIPSプロセッサ徹底活用研究
		8月号	CD-ROM付き 新世代TRONアーキテクチャT-Engine誕生

CQ出版社　〒170-8461 東京都豊島区巣鴨1-14-2　販売部 ☎(03)5395-2141　振替 00100-7-10665

組み込みLinuxをさらに便利にするしくみ

Linuxの MTD 機能を使う



海老原 祐太郎

Linux は x86系の PCだけではなく，さまざまなアーキテクチャに移植されています．いわゆるマイコン系のCPU でも Linux が動作するようになってきているので，組み込み装置用の OSとして Linux を検討している人も少なくないでしょう．組み込み Linux に取り掛かるエンジニアが最初に疑問に思う項目の一つとして，『ファイルの置き場』が挙がるのではないでしょうか．

Linux に限らず，ほとんどの UNIX 系 OS では，『ファイル』を単位とした外部記憶を使用しています．そのため，ファイルを保存する領域が必要になってきます．組み込み装置においては，このようなファイルを保存する領域（ストレージ；外部記憶装置）としてフラッシュ・メモリを使用することが多いでしょう（表1）．

表に示したように数多くの種類のフラッシュ・メモリを用いたストレージが存在します．これらのストレージ・デバイスは，CPU メイン基板と何らかのインターフェースを通じて接続されています．ことばのとおり『外部記憶装置』です．このようなフラッシュ・メモリ型外部記憶装置は大容量の NAND 型フラッシュ・メモリ＋メモリ・コントローラで構成され，仕様化されたレジスタを通してアクセスするため，メディアの交換や増設がしやすいといった利点があります．また，組み込み装置と PC 間のデータの受け渡しを考慮し，FAT ファイル・システム・フォーマットが多く使用されます．

一方で，従来の ROM と同じく，プログラム格納/読み出しを目的としてプリント基板上に実装され CPU と直結するタイプの ROM にも，フラッシュ・メモリが多用されています．ISP（イン・システム・プログラミング）や出荷後のプログラムのバージョン・アップが可能になる，未使用領域にパラメータを保存できるといったことが可能になったのも，フラッシュ・メモリのメリットでしょう．

本稿で解説する Linux の MTD（Memory Technology Device）とは，前記のように CPU に直結してメモリ・マップの中に存在する各種のメモリを，Linux 上から『デバイス』として扱うためのソフトウェア・レイアです．MTD を活用することにより，CPU メモリ・マップ上のフラッシュ・メモリや SRAM といったメモリ・チップ上にファイル・システムを構築したり，あるいは従来のマイコンと同じように SRAM 上にワーク・エリアを持たせるといった使い方が可能になります．

実装済みメモリは，物理的に取り外すことが不可能なので他機種とのメディア交換を意識しなくて済みます．そのため，ファイル・システム形式は FAT に限定されることはなく，独自の形式も含めて多種類の中から選択することができます．

MTD の特徴

Linux の MTD 機能は数年前から存在していましたが，残念ながら筆者がはじめて MTD に取り組んだときには具体的な使用方法を記述したドキュメントや Web サイトは発見できませんでした．

本稿では，そのときに筆者が独自に解析した事項を元に MTD の解説をします．残念ながらすべてのプラットホーム，すべてのメモリ・チップを網羅することはできないので，表2，写真1に示す組み込み Linux 対応ボードを動作サンプルにして解説を行います．

写真1に示した組み込み Linux 対応ボード CAT709では，8M バイトのフラッシュ・メモリと，512K バイトの SRAM が実装されています．Linux の MTD 機能を使って，この2種類のメモリをデバイスとして認識させ，ファイル・フォーマットしてルート・ファイル・システムとしてマウントしています．こうすることで，CF ソケットがフリーになり，無線 LAN カードや PHS 通信カードなどを利用することができるようになります．

MTD レイアの構成

図1に，Linux2.4カーネルに含まれる MTD のレイアを示します．ただし，この図に記載のレイア（チップ・ドライバ層，

表1　フラッシュ・メモリのストレージの例

名　称	インターフェース
CompactFlash	パラレル・バス接続， ハードディスク互換 I/F
SD メモリ・カード	シリアル接続
USB フラッシュ・メモリ	USB マス・ストレージ標準 I/F

表2　CAT709のおもな仕様

CPU	SH7709S(SH-3) 117MHz
メイン・メモリ	SDRAM 32Mバイト
フラッシュ・メモリ	8Mバイト
バックアップ・メモリ	SRAM 512Kバイト
インターフェース	100Base-TX Ethernet×1 シリアル×3 DIO
その他	CFソケット 時計IC

表3　チップ・ドライバ

amd_flash.c	AMD compatible flash chips(non-CFI)
cfi_cmdset_0001.c	CFI Intel Extended Vendor Command Set
cfi_cmdset_0002.c	CFI AMD & Fujitsu Standard Vendor Command Set
cfi_cmdset_0020.c	CFI ST Advanced Architecture Command Set
jedec.c	非CFI旧タイプJEDECフラッシュ・インターフェース
map_ram.c	単純RAM
map_rom.c	単純ROM
sharp.c	pre-CFI Sharp flash chips
blkmtd.c	IDEやSCSIブロック・デバイスをMTDと仮想化させるドライバ
doc1000.c	Disk-On-Chip 1000?
doc2000.c	Disk-On-Chip 2000 and Millennium
doc2001.c	Disk-On-Chip Millennium
lart.c	28F160F3フラッシュ・メモリ(non-CFI) on LART
ms02-nv.c	DEC MS02-NV NVRAMモジュール・ドライバ
mtdram.c	test mtd device
pmc551.c	PMC551 PCI Mezzanine RAM Device
slram.c	メイン・メモリの未使用RAM領域を使用するドライバ

マップ層など)には一般化された名称がなさそうだったので筆者が独自に命名しました．LinuxのMTDの各レイアは比較的きれいに分離され，理解しやすくなっています．この図を元にハードウェアに近い下のレイアから順に解説します．

● チップ・ドライバ層

SRAM，フラッシュ・メモリといったメモリ・チップに対する入出力を扱うレイアです．このレイアには標準でRAM，ROM，CFI対応フラッシュ・メモリ，JEDEC EEPROM，各種NANDフラッシュ・メモリといったメモリ・チップの入出力ドライバが用意されています(表3)．

各チップの入出力ドライバは，メモリ・チップに対する

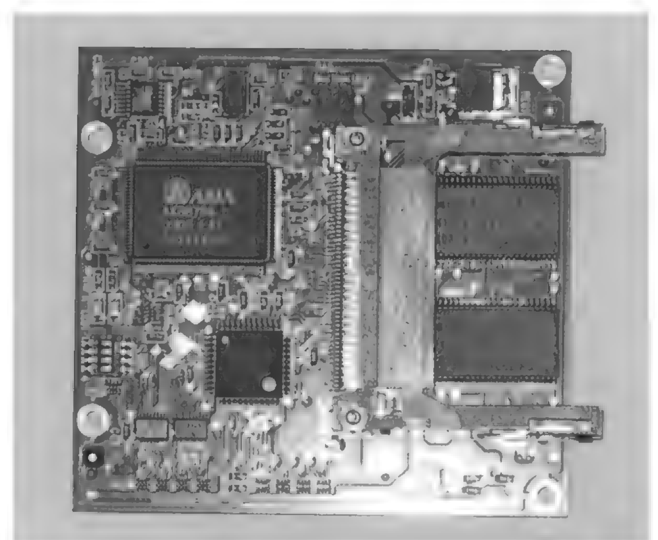


写真1　組み込みLinux対応ボード CAT709

MTD(memory technology device)を使用すると組み込み向けLinuxにおいてメモリ・チップの取り扱いが断然楽になる．チップの種類ごとに入出力ドライバが用意され，レイア構造が比較的きれいに分離している(ただし，例外や抜け道もある)

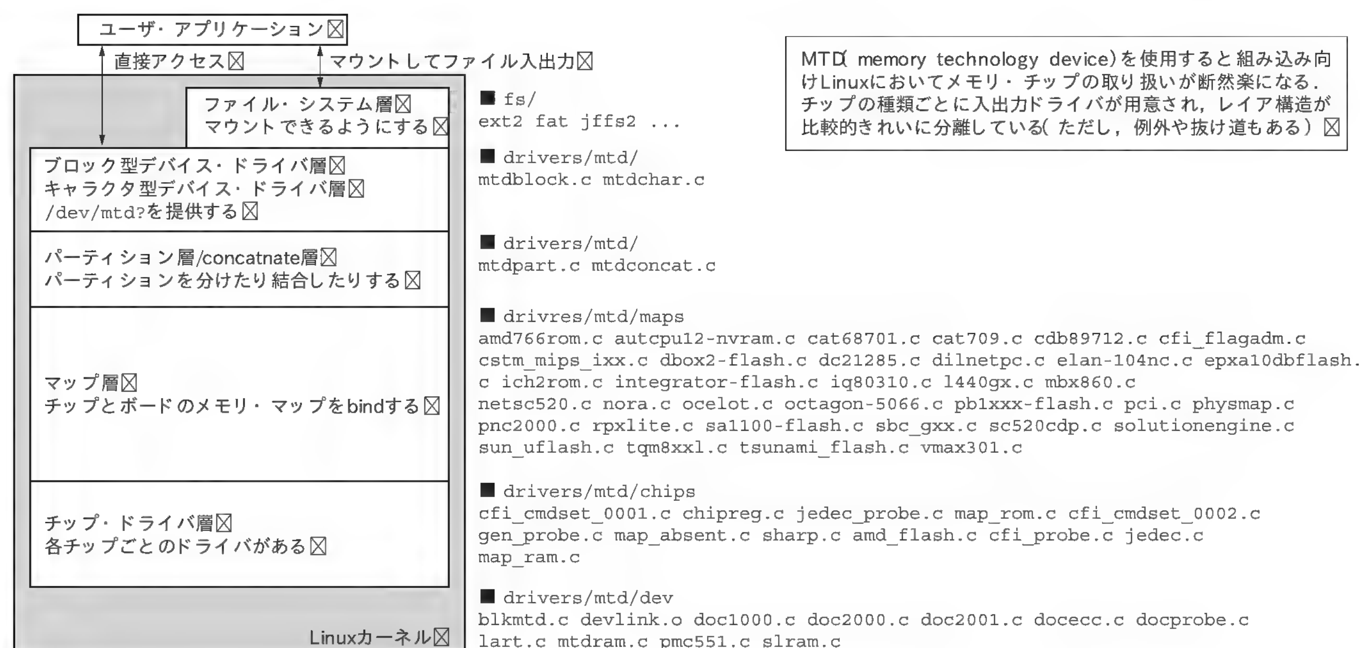


図1　Linux 2.4mtdの構成

erase()，read()，write()，sync()といったメソッドを提供します．どのようなメモリ・チップの入出力ドライバが用意されているかは，カーネルのdrivers/mtd/chips/，drivers/mtd/devicesディレクトリに含まれる各ソース・コード名称から，およそ見当がつくと思います．

ちなみに，CFIはCommon Flash Interfaceの略で，フラッシュ・メモリのベンダID，プロダクトIDをはじめ，チップ・サイズやイレース・セクタ・サイズなどの自動検出の手順を共通化したものです．

● マップ層

この層では各種チップと基板上でのメモリ・マップを組み合わせています．

先のCAT709ボードでは，CFI対応フラッシュ・メモリとSRAM（単純なSRAM）のチップ・ドライバに対して，メモリが存在するベース・アドレスを指定しています．

ボードの設計者は，このマップ層のコードを書くことでチップとメモリ・マップの対応をとることができます．逆にいえば，基板の設計者はこのマップ層のみを管理すればよいということになります．

● パーティション層，連結（concatenate）層

ハードディスクをいくつかのパーティションに分割するのと同じく，一つのメモリ・チップを複数の領域に分割するためのレイアがパーティション層です．たとえば，CAT709では図2のようにフラッシュ・メモリを三つの領域に分割しています．

連結（concatenate）層は，分割した複数のパーティションを再び連結させるためのレイアです．使用している例はあまりありません．

● キャラクタ型デバイス・ドライバ層，ブロック型デバイス・ドライバ層

MTDデバイスをIDEやSCSIといった装置と同じようにブロック型デバイス・ドライバとして上位層に認識させるためのレイアです．このレイアを通すことで，MTDデバイス上にファイル・システムを構成することができます．

● ファイル・システム層

MTDレイアの最上位に位置するのがファイル・システム層です．ほかのIDEやSCSI，FD，MOといった外部記憶装置と同様に，Linuxがもっている豊富な種類のファイル・システムを構成することができます．ファイル・システム形式の候補としてはFAT，ext2，reiserfs，そしてjffs2が考えられます．

表4にLinuxがもつ各種ファイル・システム形式の特徴を挙げてみました．このうち，先に述べたように他機種とのメディア交換や互換性を重視する場面では，事実上FAT以外の選択肢がなくなります．しかし，FATでは残念ながらユーザやパーミッションといったUNIX特有のファイル情報を保存できません．

MTDではメディアを取り外して交換するといったことは考

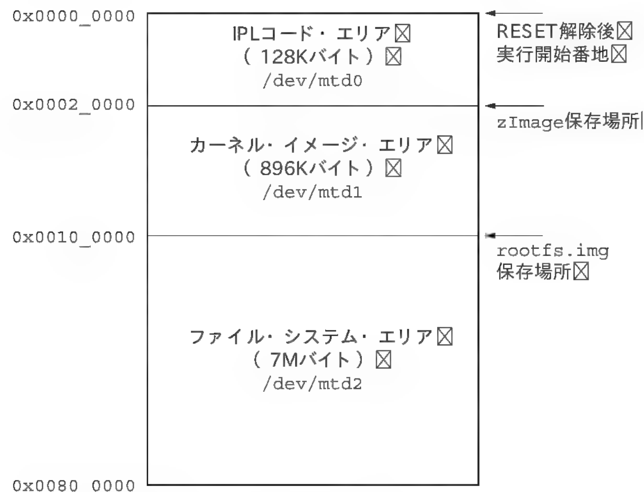


図2　フラッシュ・メモリのパーティション

表4　Linux各ファイル・システム形式の特徴

ファイル・システム	メリット	デメリット
FAT	しかけが簡単・高速 他機種と互換性がある	UNIX特有のファイル情報が保存できない． 不意の電源断に弱い． MTD上でFATを利用する意味は薄い
ext2	Linux標準ファイル・システム 高速で安定している	不意の電源断に弱い
reiserfs	ジャーナリング機能 小さなファイルが多い場面で有利	最低でも32Mバイト以上の領域を想定
jffs2	ジャーナリング機能 透過圧縮対応	MTD以外への応用が手間 比較的遅い
cramfs	完全readonly 圧縮対応	書き換え不可能

COLUMN 1

ライセンスに関して

ご存じのようにLinuxはLinus Torvalds氏が著作権を持つ著作物で，氏が定めたライセンスを無視して使用することはできません．Linuxカーネル全体はGPLで配布されています．GPLでは自身の追加部分もGPLにするという条件を守る場合に限り，追加・変更が認められています．このため，MTDに関するコードをカーネルに追加する場合は，追加部分をGPLにする義務が発生します．

広く知られていることとして，一般的なデバイス・ドライバを外部モジュールにした場合には，カーネルに対するローディングと認識されるので，GPLでなくてもよいという慣例的な認識があります（これは文書化されていない）．

ところが，本文中で述べたようにMTDに関するコードは外部モジュールにすることはできないので，カーネル内部にスタティックに追加する行為に該当します．この部分はGPL以外にする方法はありません．

リスト 1　cat709.c

```
linux-2.4.19/drivers/mtd/maps/cat709.c
#include <linux/module.h>
#include <linux/types.h>
#include <linux/kernel.h>
#include <asm/io.h>
#include <linux/mtd/mtd.h>
#include <linux/mtd/map.h>
#include <linux/mtd/partitions.h>
__u16 cat709_read16(struct map_info *map, unsigned long adr)
{
    return readw(map->map_priv_1 + adr);
}
void cat709_write16(struct map_info *map,
                                __u16 d, unsigned long adr)
{
    writew(d, map->map_priv_1 + adr);
    mb();
    return;
}
void cat709_copy_from(struct map_info *map, void *to,
                            unsigned long from, ssize_t len)
{
    memcpy_fromio(to, map->map_priv_1 + from, len);
}
void cat709_copy_to(struct map_info *map, unsigned long to,
                              const void *from, ssize_t len)
{
    memcpy_toio(map->map_priv_1 + to, from, len);
}
struct map_info cat709_flash_map = {
    name: "CAT709 Flash1",
    map_priv_1: 0xa0000000, /* base address */
    size: 0x00800000,
    buswidth: 2,
    read16: cat709_read16,
    write16: cat709_write16,
    copy_from: cat709_copy_from,
    copy_to: cat709_copy_to
};
struct map_info cat709_sram1_map = {
    name: "CAT709 SRAM",
    map_priv_1: 0xb0000000, /* base address */
    size: 512*1024,
    buswidth: 2,
    read16: cat709_read16,
    write16: cat709_write16,
    copy_from: cat709_copy_from,
    copy_to: cat709_copy_to
};
static struct mtd_partition cat709_flash_partitons[] = {
    /* IPL erea (128K byte) */
    {
         name: "ipl",
         offset: 0,
         size: 128*1024,
         mask_flags: 0, /* MTD_WRITEABLE */
    },
    /* kernel erea (1M - 128K byte) */
    {
         name: "kernel",
         offset: MTDPART_OFS_NXTBLK,
         size: (1024-128)*1024,
         mask_flags: 0, /* MTD_WRITEABLE */
    },
    /* All else is for file system */
    {
         name: "Flash FS",
         offset: MTDPART_OFS_NXTBLK,
         size: MTDPART_SIZ_FULL,
         mask_flags: 0 /* MTD_WRITEABLE */
    }
};
static struct mtd_info *flash_mtd, *sram1_mtd;
/* ========================================= */
static int __init init_cat709_maps(void)
{
    int nr_parts;
    /* Flash Map */
    flash_mtd = do_map_probe("cfi_probe", &cat709_flash_map);
    if (!flash_mtd) {
         /* Not there. */
         printk(KERN_NOTICE "cat709.c: Flash chips1 not
                                                  detected.¥n");
         return -ENXIO;
    }
    flash_mtd->module = THIS_MODULE;
    sram1_mtd = do_map_probe("map_ram", &cat709_sram1_map);
    if (!sram1_mtd) {
         printk(KERN_NOTICE "cat709.c: SRAM1 not detected.¥n");
         return -ENXIO;
    }
    sram1_mtd->module = THIS_MODULE;
    /* register /dev/mtd? devices */
    nr_parts = sizeof(cat709_flash_partitons)/sizeof(struct
    mtd_partition);
    add_mtd_partitions(flash_mtd, cat709_flash_partitons,
                                                   nr_parts);
    add_mtd_device(sram1_mtd);
    return 0;
}
static void __exit cleanup_cat709_maps(void)
{
    if(flash_mtd) {
         del_mtd_partitions(flash_mtd);
         map_destroy(flash_mtd);
    }
    if(sram1_mtd) {
         del_mtd_device(sram1_mtd);
         map_destroy(sram1_mtd);
    }
}
module_init(init_cat709_maps);
module_exit(cleanup_cat709_maps);
```

慮しないのでFATを選ぶ理由は薄くなります．従来からのlinuxの標準的なファイル・システム形式としてext2が一番普及しています．ext2はそこそこ高速で安定していて信頼性もあり，良い意味で枯れているといえます．難点としては，不意の電源断に弱い，圧縮が効かないといったことがあげられます[注1]．

さて，jffs2形式はジャーナリングに対応し，透過的な圧縮伸張機能を備え，書き換えも可能という優れたファイル・システム形式です．CAT709ではjffs2形式を利用しました．

cramfsは完全readonlyなファイル・システムで，書き換えはまったく不可能ですがjffs2よりも管理構造が単純な分，データそのものは多く入ります．書き換え不可能な点はデメリットと感じられますが，逆に完全ROMディスクで運転可能な装置では書き換え不可能な点がメリットになるかもしれません．読み込みも高速です．jffs2よりも圧縮率が高いので小さな領域にファイルをたくさん詰め込むときには有利です．ただしファイル・サイズ16Mバイト以下，デバイス領域256M以下という制限があります[注2]．CAT709ではcramfsもサポートしています．

ファイル・システムを経由することにより，アプリケーションからはデバイス上にファイルを置くことができるようになります．扱いにくかったフラッシュ・メモリも高機能なファイル・ストレージ領域として扱うことができるようになります．一方で，たとえばSRAMなど，ファイル置き場として使うより，昔のマイコンと同様に，アドレスを直に決めてワーク・エ

注1：圧縮ext2形式を開発しているプロジェクトもある．
注2：このため，CD-ROMにcramfsイメージを保存して，より多くのファイルを保存するといった使い方ができないのが残念．

表5　struct map_info 構造体

name	識別子
map_priv_1	ベース物理アドレス
size	メモリのサイズ(バイト単位)
buswidth	バス幅(バイト単位)
read16	読み込みメソッド
write16	書き込みメソッド
copy_from	連続読み込みメソッド
copy_to	連続書き込みメソッド

表6　struct mtd_partation 構造体

name	パーティション名
offset	オフセット・アドレス(バイト単位)
size	パーティション・サイズ(バイト単位)
mask_flags	ビット・マスク(ライト許可など)

表7　重要なカーネル関数のリファレンス

struct mtd_info *do_map_probe(char *name, struct map_info *map)		
機　能	mapのデバイスを検査する	
引き数	char*　デバイス名 "jedec_probe" "cfi_probe" "map_rom" "map_ram" など struct map_info*	 マップ情報構造体
戻り値	struct mtd_info*	mtd_info 構造体 失敗時 NULL
int add_mtd_partitions(struct mtd_info *master, struct mtd_partition *parts, int nbparts)		
機　能	mtdパーティションを追加する	
説　明	カーネル・グローバル mtd_partitions リストに mtd_part 構造体(slave)をリストでつなげていく．mtd_part 構造体の中に mtd_info 構造体がある．	
引き数	struct mtd_info* struct mtd_partition* int　nbparts	追加する mtd 構造体 パーティション情報構造体 パーティション数
戻り値	int	正常終了時 0，失敗時 負

リア的に使用したい場面もあるかもしれません．その場合はファイル・システムを経由せずに直接ブロック・デバイスにアクセスすることで固定サイズのワーク・エリア的な使い方も可能です．

プログラム・コードの実際

MTDレイアの項で述べたように，LinuxMTDではチップ・ドライバ層として多種多様なメモリ・チップの入出力関数が用意されているので，独自のメモリ・チップを使うとき以外はチップ・ドライバ層のコードを書くことはないと思われます．メモリ・チップとメモリ・マップの関連付けを行うのはマップ層になるので，マップ層とパーティション層のプログラム・コードはボードごとに書く必要があります．ここでは前述したCAT709のマップ層のコードを例に解説します(cat709.c，リスト1)．

構造体struct map_info(表5)は，メモリ・マップを管理しています．構造体struct mtd_partation(表6)は，メモリ・チップ内のパーティション分けを管理しています．構造体struct mtd_infoは，mtdデバイスそのものになります．mtd_infoのメンバは，直接参照することはありません．具体的な処理の流れはプログラム・コードを参照してください．

重要なカーネル関数のリファレンスを表7に示します．

```
$ tar xzvf linux-2.4.24.tar.gz
$ cd linux-2.4.24
$ make menuconfig  (もしくは make xconfig)


Memory Technology Devices (MTD)  --->
  <*> Memory Technology Device (MTD) support
      mtd機能を使う
  <*>  MTD partitioning support
       MTDパーテーション・サポート
  <*>  Direct char device access to MTD devices
       MTDデバイスへのキャラクタ型デバイス・ドライバ
  <*>  Caching block device access to MTD devices
       MTDデバイスへのブロック型デバイス・ドライバ
 RAM/ROM/Flash chip drivers  --->
   <*>  Detect flash chips by Common Flash Interface
                                           (CFI) probe
        CFIフラッシュ・メモリ
   <*>  Support for Intel/Sharp flash chips
        Intel/Sharp形式フラッシュ・メモリ・チップ
   <*>  Support for AMD/Fujitsu flash chips
        AMD/FUjitsu形式フラッシュ・メモリ・チップ
   <*>  Support for RAM chips in bus mapping
        単純RAMメモリ・チップ
File systems  --->
  <*> Journalling Flash File System v2 (JFFS2) support
      jffs2ファイル・システム・サポート
  <*> Compressed ROM file system support
      cramfsファイル・システム・サポート
```

図3　カーネルのコンフィグレーション

カーネル・コンフィグレーション

Linuxカーネルをコンフィグレーションする際にmtdサブシステムを組み込めばMTD機能が使えるようになります．Linux2.4カーネルのコンフィグレーションを図3に示します．

Linuxカーネル・コンフィグレーションでは，各サブシステムをカーネルにスタティックに組み込むか，モジュール・ファイルとして後からロードするかを選べます．

MTDに関しては，モジュールも選択できますが，スタティック組み込みを選択してください．MTDがモジュールになっていると，必要なモジュールがメモリ・チップの中にファイルとして存在し，ロードできないという矛盾が生じます．コンフィグレーションが終わったら，

```
$ make dep
```

COLUMN 2

Linux2.6では

Linux2.6では，struct map_info構造体のメンバ構成が若干変更になったもようです．Linux2.4ではメモリ・チップの物理ベース・アドレスを，.map_priv_1メンバ変数に保存していましたが，Linux2.6では，.phys（物理番地）.virt（仮想番地）それぞれのメンバ変数に記録するようになりました．そのほかは特に変更はないようです．cat709.cの一部から抜粋して示します．

```
cat709_flash_map.phys = 0x0;
                              // 0xa0000000;
cat709_flash_map.size = 0x00800000;
cat709_flash_map.buswidth = 2;
cat709_flash_map.virt
        = P2SEGADDR(cat709_flash_map.phys);
simple_map_init(&cat709_flash_map);
```

```
$ make zImage
```

としてカーネルをメイクします．

コンパイルが終わると，arch/sh/boot/zImageファイルができあがります．これがカーネル・イメージになります．

● パーティションを分けた理由

CAT709では表8に示すようにフラッシュ・メモリを三つの領域に分割しています．

/dev/mtd0はフラッシュ・メモリの先頭から128Kバイトまでとしました．フラッシュ・メモリの先頭は，CPUのリセット解除直後のコード・フェッチ番地になっています．この領域にブート・コードが書き込まれています．

ブート・コードの仕事は，ハードウェアの初期化とカーネルのロードです．

さて，カーネルですが，普通のPCの場合はハードディスクの中にファイルとして保存されています．しかし，組み込み機器においては，通常のPCとは異なりハードウェアやメモリ・マップの設計は設計者の自由に任されているので，PCと同じように無理にファイルとして保存させる必要はありません．ファイルとしてファイル・システムの中に保存してしまうと，ブート・プログラムにはファイル・システムを理解してファイルを探し出すルーチンを仕組まなければなりません．

CAT709ではカーネル・イメージを保存する専用の領域を設け，その領域の中にカーネルを保存しています．/dev/mtd1領域がそれにあたり，ブート・プログラムは単純に固定されたROM番地からカーネル・イメージをロードしています．

/dev/mtd2は，ファイル・システム領域です．Linux起動後の/（ルート）ファイル・システムがここにあたります．

表8　CAT709のパーティション

デバイス名称	サイズ	用　途
/dev/mtd0	128Kバイト	ブート・コード
/dev/mtd1	896Kバイト	カーネル・イメージ保存
/dev/mtd2	7Mバイト	ファイル・システム領域

ファイル・システム・イメージ

/dev/mtd2に書き込むファイル・システム・イメージを作ります．ファイル・システム・イメージの作り方は，ファイル・システム形式によって若干異なりますが，およその流れとしては次のようになります．

1) 開発機上で，ディレクトリ上にファイルをすべて並べる
2) イメージ・ファイル・メイク・ツールでイメージ・ファイルを作る
3) ICEなどの開発機材やROMライタなどでフラッシュ・メモリに書き込む

各ファイル・システム形式の作業例を示します．

イメージ・ファイルのメイク・ツールのインストールはディストリビューションによって異なりますが，筆者が利用しているDebianでは非常に簡単です．

```
# apt-get install mtd-tools    jff2のツール
# apt-get install mkcramfs     cramfsのツール
```

● ファイル・システム・イメージの構成方法

●jffs2の場合

```
$ su -   rootユーザでないと扱えないファイルもあるので
# mkdir target   targetディレクトリに
                 すべてのファイルを集める
# echo "hello" >test.txt   最低一つのファイルがな
                           いとエラーになるので
# cd ../
# mkfs.jffs2 -r target/ -p -l -e 0x10000
                                   -o rootfs.img
```

オプションの意味は，

-r ディレクトリを指定
-p パディング．未使用領域を0xffで埋める
　（manには0x00で埋めると書かれているが0xffの誤り）
-l リトル・エンディアン
-e 消去ブロック・サイズ
-o 出力ファイル

```
# ls -l rootfs.img
-rw-r--r--   1 root root 65536
                        x月 xx 16:05 rootfs.img
```

消去ブロック・サイズを64Kバイトとしたので，最小のイメージ・ファイルが64Kバイトとなりました．

● cramfsの場合

```
$ su -    rootユーザでないと扱えないファイルもあるので
# mkdir target  targetディレクトリに
                すべてのファイルを集める
# echo "hello" >test.txt  最低一つのファイルがな
                          いとエラーになるので
# cd ../
# mkcramfs target/ rootfs.img
```

● ext2fsの場合

```
$ su -    rootユーザでないと扱えないファイルもあるので
# dd if=/dev/zero of=rootfs.img
                              bs=1M count=7
      インプット・ファイルを/dev/zeroとした．連
      続して0が読み出せる特殊デバイス
      アウトプット・ファイルをrootfs.imgとして，
      7Mバイトの0x00で埋まったファイルを作る
# mkfs.ext2  rootfs.img
      rootfs.imgをext2形式でフォーマットする
mke2fs 1.27 (8-Mar-2002)
rootfs.img is not a block special device.
Proceed anyway? (y,n) y
      良いかどうか聞かれるのでyと答える
# mount -o loop rootfs.img /mnt/
      rootfs.imgを/mntディレクトリにループ・
      バック・マウントする
# cp test.txt /mnt
      必要なファイルをすべて/mntにコピーする
# umount /mnt
```

これで7Mバイト・サイズのrootfs.imgができあがります．

いずれの場合でも，でき上がったrootfs.imgをファイル・システムとして使用するフラッシュ・メモリの領域に書き込むことになります．

カーネル・ブート

カーネルのブートは固定番地に保存されているカーネル・イメージを，RAM上のロード・アドレスにコピーしてエントリ・ポイントにジャンプすればよいので，ローダの作成はいたって簡単です．sh-linuxではカーネルのロード・アドレスのデフォルト値は表9のようになっています．したがって，リスト2のようなローダでカーネルをロードしています．

ルート・ファイル・システムのマウント

カーネルは起動直後に/(ルート)ファイル・システムをマウントしようとします．

カーネル起動パラメータのroot=行でカーネルに対して/(ルート)デバイスを知らせることができます．root=パラメータには16進数4桁で，/デバイスのメジャー番号とマイナ番号を記述します．デバイスのメジャー番号マイナ番号の一覧はカーネル・ソースのDocumentation/devices.txtにすべて記述されています．代表的な番号を表10に示します．

CAT709では，/dev/mtd2が/(ルート)ファイル・システムになります．メジャー番号31，マイナ番号02なのでカーネルに対して起動パラメータを16進数記述でroot=1f02として与えます．

表9　sh-linuxカーネルの絶対番地(デフォルト)

zImageロード・アドレス	0x8c210000
カーネル・エントリ・ポイント	0x8c210000

表10　代表的なブロック・デバイスの番号

メジャー番号	マイナ番号	デバイス名	デバイス
3	1	/dev/hda1	第1IDE第1パーテーション
3	2	/dev/hda2	第1IDE第2パーテーション
22	1	/dev/hdc1	第2IDE第1パーテーション
22	2	/dev/hdc2	第2IDE第1パーテーション
31	0	/dev/mtd0	最初のmtd領域
31	1	/dev/mtd1	2番目mtd領域
31	2	/dev/mtd2	3番目mtd領域

リスト2　カーネル・ローダ

```
/* kernel image */
printf("loading linux¥n¥n");
for(i=0; i<0x100000-0x20000; i++){ /* 1M-128K byte */
  c = *(unsigned char*)(KERNEL + i);
  *(unsigned char*)(CONFIG_RAM_BOOT + 0x10000 +i) = c;
}

asm volatile ("jmp @r0; nop"
          : : "z" (CONFIG_RAM_BOOT + 0x10000));

// CONFIG_RAM_BOOT+0x10000 は 0x8c210000のこと
// KERNELは /dev/mtd1 の先頭番地のこと
```

mtd領域の直接アクセス

Linuxが起動してしまえば，/dev/mtd*はそれぞれのサイズをもったブロック・デバイスになります．前述したように，/dev/mtd0はブート・コードそのものを示しているので，

```
# cp sh-stub.bin /dev/mtd0
```

(sh-stub.binはブート・コードのバイナリ・イメージ)とすれば，ブート・プログラムの書き換えができます．同じく，カーネル・イメージであるzImageファイルを，

```
# cp zImage /dev/mtd1
```

として/dev/mtd1領域に書き込むことでカーネルのアップ・デートができます．

UNIXにおいては，すべてがファイルであるという認識がありますが，LinuxのMTDを用いることでメモリ領域すべてを

リスト 3　アプリケーション・プログラムの例

```
/*
 SRAM mmap test program
 2004-06-26 SiliconLinux y.ebihara
*/

#define SRAMSIZE (4*1024) /* SRAM 4Kbyte */
#include <stdio.h>
#include <fcntl.h>
#include <sys/mman.h>

int main(){
 int fd;
 unsigned char *sram;
 printf("this is sram test program No.1
                                    2004-06-26 Y.Ebihara¥n");

 /* SRAM デバイス /dev/mtd3 を open する */
 fd=open("/dev/mtd3",O_RDWR);
 if(fd<=0){
  perror("");
  exit(1);
 }

 /* fd を使って mmap する(ポインタが戻る)*/
 sram=mmap(0,SRAMSIZE,PROT_READ|PROT_WRITE,MAP_SHARED,fd,0);
 if(sram<0){
  perror("");
  exit(1);
 }
 printf("SRAM mmaped at %p¥n",sram);

 /* そのポインタを使ってアクセスする */
 sram[0]=0x12;
 sram[1]=0x34;
 sram[2]=0x56;
 sram[3]=0x78;
 /* SRAMの先頭4バイトに書き込まれました */

 /* SRAM と同期をとる(ハードへの書き込み)*/
 msync(sram,SRAMSIZE,MS_SYNC);

 /* mmap を閉じる */
 munmap(sram,SRAMSIZE);

 /* デバイスを閉じる */
 close(fd);
}
```

```
# ./sramtest
this is sram test program No.1 2004-06-26 Y.Ebihara
SRAM mmaped at 0x2956a000

# hex /dev/mtd3
0x00000000: 12 34 56 78 00 00 00 00 - 00 00 00 00 00 00 00 00 R4Vx@@@@@@@@@@@@
0x00000010: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 @@@@@@@@@@@@@@@@
```

図4
アプリケーションの実行結果

ファイルとして扱うこともできるようになります．

SRAM

CAT709では，三つのフラッシュ・メモリ領域に加えて4番目の領域である/dev/mtd3がバッテリ・バックアップされた512KバイトのSRAMになっています．/dev/mtd3は512Kバイトのメディアとして使用できるので，

1) フォーマットしてマウントする
2) ワーク・エリアとして使用する

という二つの利用方法が使えます．1番目の使い方は普通のブロック・デバイスとまったく同じです．基本的な使い方は次のとおりです．

```
# mkdir /sram　　マウント・ポジションの作成
# mkfs.minix /dev/mtd3　　フォーマット
# mount -t minix /dev/mtd3 /sram　　マウント
```

SRAMをワーク・エリアとして使用するおもしろい方法があります．図1で示すところの，ファイル・システムを経由せず直接アクセスする方法です．

UNIXにはmmap()というおもしろいしくみがあります．mmap()は「ファイルをmallocする」という感覚に近いと表現できます．ファイルをmmap()すると仮想メモリのポインタが戻り，このポインタ上にファイルが張り付いているように見えます．通常のファイルの代わりにデバイス/dev/mtd3をmmap()すれば，戻りポインタにSRAMが張り付いているように見えるわけです．アプリケーション・プログラムの例をリスト3に示します．

実行結果は図4のようになります．

通常，このようなmmap()を実現させるためには，デバイス・ドライバ内に自分でmmap()デバイス・メソッドを書かなければなりません．LinuxMTDを利用すると，mmap()のしくみはすでに用意されているので，ボード設計者は車輪の再開発を行わなくても済みます．

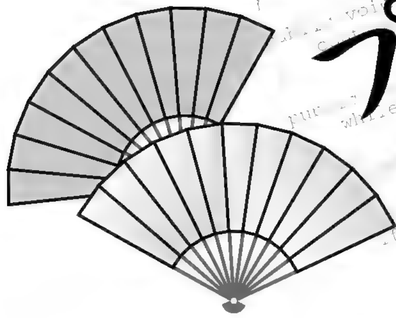
＊　　＊

以上，駆け足でしたがLinuxのMTD機能に関して筆者が調べた範囲でまとめてみました．見てきたようにMTDは組み込み機器において非常に便利な機能です．

MTDに限らず，Linuxの内部構成は日進月歩で進化しています．構造体やカーネル関数も変化していくので最新の情報を集めてください．

参考Webサイト
1) http://www.si-linux.com/product/cat709/

えびはら・ゆうたろう　シリコンリナックス(株)

プログラミングの要

宮坂 電人

第15回

グラフ──頂点と辺からなるデータ構造

今回は「グラフ(graph)」について説明します．グラフは「頂点(vertex)」と称する点と，「辺(edge)」と称する線で構成された図形です．ただし，グラフが取り扱うのは幾何学的な分野だけに限定されず，さまざまな応用例が考えられます．

グラフとは何か

グラフにとって重要なのは，「頂点がどの位置にあるか？」とか，「辺がどのような形状として描かれているか？」ではなく，「どの頂点とどの頂点に辺がつながっているか？」とか，「どの頂点からどの頂点に向かってつながっているか？」という接続関係です(図1)．図で示されている二つのグラフはともに六つの頂点があり，それぞれの頂点から二つの辺が隣の頂点につながる接続関係をもっています．片方は正六角形で，もう片方は臼の形状なので幾何学的には別の図形ですが，グラフという観点では同じものだと考えられます．

● グラフにおける用語

グラフにはさまざまな専門用語があり，それがグラフをわかりにくくする一因となっています．ここでは必要最低限度の用語を簡潔に説明しておくにとどめます．

▶ 連結

ある頂点と別の頂点に辺がひかれている状態を「連結されている(connected)」と称します．すべての頂点で，必ずほかの頂点にいくつかの辺を経由してたどっていくことが可能なグラフを「連結グラフ」と称します．たどれない場合はどこかで分断されている状況ですが，このときグラフは複数の「部分グラフ(あるいは『連結部分グラフ』)」から構成されていると称します．

▶ 閉路

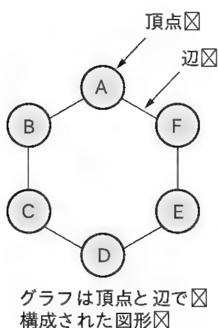
ある頂点から辺をたどっていくと，元の頂点に戻る状態がある場合，「閉路(cycle)」があると称します．閉路があるグラフは頂点をたどっていくとき無限ループにはまりこむ危険性があるので，プログラムで処理するときは，それぞれの頂点に巡回したかを示すフラグを持たせる必要があります．

閉路のない連結グラフは「ツリー(tree)」と称します[注1]．四角形の特殊化した形態として正方形がありますが，この対応でとらえるならば，グラフは四角形に相当し，ツリーは正方形に相当するでしょう．

● 辺の取り扱い

単純に頂点どうしを辺で結んでいるだけで，片方の頂点から，もう片方の頂点にたどれ，その逆方向もたどれるグラフを「無向グラフ(undirected graph)」と称します．それに対し，辺に方向を持たせ一方通行しかできない状況もありうるグラフを「有向グラフ(directed graph)」と称します(図2)．

単に頂点を結びつけるという意味だけでなく，なんらかの価



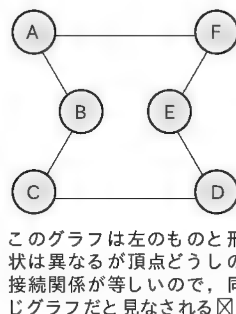
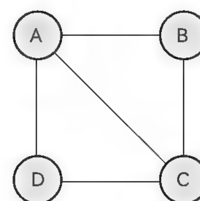
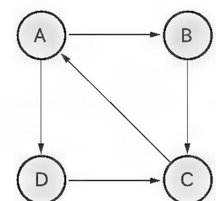


図1 グラフ



辺に方向はなく，辺がつながる頂点どうしで移動ができる．
上のグラフでは，
A→B，B→A，A→C，C→A，A→D，D→A，D→C，C→D，B→C，C→B
の10通りの移動ができる

(a) 無向グラフ

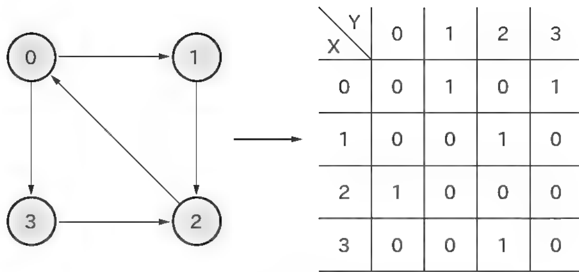


辺に方向があり，
一方通行になっている．
上のグラフでは，
A→B，A→D，D→C，B→C，C→A
の5通りの移動のみ

(b) 有向グラフ

図2 有向グラフと無向グラフ

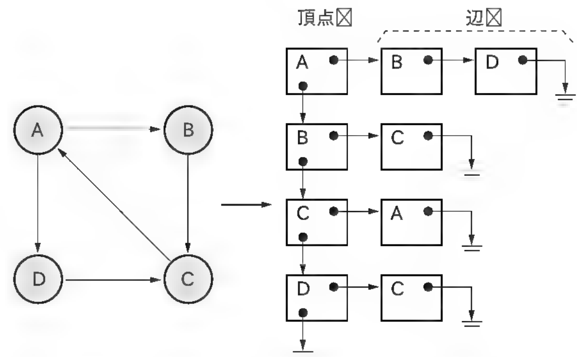
注1：互いが連結されていないツリーが複数あるグラフのことを「森(forest)」あるいは「林」と称することがある．



X＼Y	0	1	2	3
0	0	1	0	1
1	0	0	1	0
2	1	0	0	0
3	0	0	1	0

隣接行列表現は辺の状態を2次元配列で表現する☒

図3　隣接行列表現



隣接リスト表現は辺の状態を連結リストで表現する☒

図4　隣接リスト表現

値を辺に持たせたいことがあります．具体的には頂点どうしの距離や，ある頂点から隣の頂点に行くまでにかかる時間や費用（cost）などです．これらを数値にしたものを「重み（weight）」と称し，辺に重みを付加したグラフを「重みつきグラフ（weighted graph）」と称します[注2]．

プログラムでのグラフの取り扱い

グラフに関する講義では，この後，退屈な原理の説明や証明問題がえんえんと続くところです．それはグラフ理論はプログラミングよりも数学としてとらえられる局面が多いからです．しかし「プログラミングの要」としてはあまりプログラミングに関係しない話題は遠回りになると考えられるので，ただちにグラフのプログラミングに取りかかりましょう．

最初に悩むポイントとして，グラフをどのようなデータ形式（表現）として持つべきかがあります．おもに以下の2パターンが採用されているようです．

1）隣接行列表現（adjacency matrix representation）

頂点がN個あった場合，`a[N][N];`という2次元配列を用意します．この配列は，ある頂点から別の頂点に対して辺がひかれているかを表現します（図3）．たとえば，頂点Xと頂点Yがあったときに，2次元配列を論理型にしておくと，`a[X][Y]`が`false`なら辺がひかれていない状態で，`true`なら辺がひかれている状態とみなします．無向グラフなら`a[Y][X]`も同じ値になるはずですが，有向グラフの場合は同じ値になる保証はありません．重みつきグラフでは2次元配列を論理型ではなく，数値型にしておくと，辺の重みが記録できます．

2）隣接リスト表現（adjacency list representation）

ある頂点から別の頂点への辺が存在するなら，その辺を連結リストに記録します（図4）．2次元配列を使用する方法と比較すると回りくどいような印象がありますが，頂点の個数を増やしたときに動的に連結リストを拡大できること，辺の数が少ない場合[注3]にはメモリの使用量が少なくてすむという利点があります．頂点の個数が多い場合は連結リストではなくハッシュ・テーブルやSTLの`set`や`map`のような検索効率の良いデータ構造も検討したほうがいいでしょう．

いずれの表現を採用しても有向グラフと無向グラフの両方で使用できるので便利です．それぞれの表現には一長一短があり，どちらが良いかは一概にはいえません．しかし，本連載で参考にしている「Mastering Algorithms with C」[注4]では隣接リスト表現を採用しているので，本連載でもこの表現を採用します．

● 頂点の表現

頂点にどのような情報を盛り込むかはあらかじめ決定できません．無理やり決定してプログラムに盛り込んでしまうと後から状況が変化したときにプログラムを再利用できず，いわゆる「車輪の再発明」になってしまいます．とはいうものの，グラフで頂点を扱うときに共通する処理として，最低，

1）ある頂点が示された頂点の表現と一致するかの判断
2）グラフを巡回するときに，ある頂点が巡回ずみであるかの判断

が必要になります．前者はC++を使った場合，「`==`」を演算子オーバロードすることでも実現できますが，この方法は一歩まちがえると，ある頂点と「同一」の頂点なのか「同値」の頂点なのかの判断があいまいになるというC++ならではの落とし穴にはまる危険性があります．

たとえば，鉄道をグラフに見立て，頂点に駅の名前をつけたとしましょう．ある頂点が「新宿」だとしても，これがJRなのか私鉄なのかを区別していないと，別の鉄道会社の駅なのに同じ頂点だとまちがえる危険性があります．また，その頂点クラスがプログラムの別の箇所からの要求で「`==`」を独自に演算子オーバロードしていた場合，その記述を書き換えられないというジレンマに陥ることがあります．どちらにしても演算子オーバロードは使いにくいということです．そこで，頂点の同値判定として一つの独立したメンバ関数（`equalVertex`）を設置し，

注2：重みつきグラフが有向グラフである場合，「ネットワーク（network）」と称することがある．
注3：辺が少ないグラフを「疎グラフ（sparse graph）」と称し，その反対に辺が多いグラフを「密グラフ（dense graph）」と称することがある．
注4：`http://www.oreilly.com/catalog/masteralgoc/`を参照．

頂点はこのメンバ関数を上書きする前提とします．

次に，その頂点が巡回ずみであるかの判断は論理型のメンバ変数を一つ用意するだけでいいでしょう．trueならすでに巡回しており，falseならまだ巡回していないと判断します．じつは巡回の判断を論理型ではなく3値にするという考えもありますが注5，必須であるかどうかは筆者は個人的には疑問に思います．3値にする理由や意義については次回説明することにし，ここでは2値（論理型）を採用します．

以上の考えを実現できる頂点の基本型としてリスト1のようなVertex型を用意します．グラフを利用するプログラムではVertex型を継承した独自の型を各自が用意するという考えを採用します．

● 辺の表現

辺も頂点と同様，あらかじめどのような情報を盛り込むかを決定できません．しかし，たいていの場合はどの頂点とつながっているかの情報さえあればいいので，つながっている頂点へのポインタだけを用意すればいいでしょう．頂点がどのような型になるかは，あらかじめ決定できないので，テンプレートを使ってリスト2のようなEdge型を用意します．

重みつきグラフの場合は辺に重みを表現するメンバ変数を追加する必要がありますが，それはEdge型を継承した独自の型を用意して，そこに記述するとよいでしょう．

● グラフの表現

隣接リスト表現をする場合，一つの頂点と，そこから別の頂点につなげている複数の辺をひとまとめにした構造体（struct AdjCell）を連結リストにします．複数の辺も連結リスト（EdgeList）とします．リスト3のような型宣言とメンバ変数になります．

ここで悩んだのは頂点の実体をどう確保するかという点でした．あらかじめ外部で頂点を確保しておき，Graphクラスは頂点へのポインタのみを記録するという考えでもまちがいではないのですが，グラフを処理するときに頂点の内容を書き換えてしまうことや，頂点が動的に確保されていた場合，すでに解放された頂点をうっかりアクセスしてしまう事故が予想されます．それらを予防する意味で，Graphクラス内部で頂点を動的に確保しています．そのため，Graphオブジェクトが解放されたときは動的に確保した頂点も解放する必要があります．これはデストラクタの処理として実装します．以上の配慮も含めてリスト4のようなコンストラクタとデストラクタを実装します．

一般的なオブジェクト指向の定石では，内部の実装を意識させるアクセスはあまり良くないスタイルとされています．しかし，ここでは固いことをいわずにそのまま内部をアクセスさせる手段をリスト5のように用意します．ただし，将来的に隣接情報の保持手段を変更したり，より効率の良いデータ構造に変更したときに支障が出てくる可能性があります．どのように対

注5：「Mastering Algorithms with C」ではこの考えを採用している．

リスト1　頂点の基本型

```
struct Vertex {
    bool visited; //訪問していればtrue
    //同値判定
    virtual bool equalVertex(const Vertex* iVertex) const = 0;
    virtual ~Vertex(){ /*(empty)*/ }
};
```

リスト2　辺の基本型

```
template <class VertexT>
struct Edge {
    VertexT* endVertex; //終点
};
```

リスト3　Graphクラス（型宣言とメンバ変数）

```
template <class VertexT,class EdgeT>
class Graph {
public:
    typedef std::list<EdgeT> EdgeList; //辺の集合の型
    struct AdjCell {            //隣接情報（始点と終点の集合（一つ分））の型
        VertexT* startVertex; //始点（一つ）
        EdgeList edges;       //辺の集合（複数）
    };
    //隣接情報（始点と終点の集合）（複数）の型
    typedef std::list<AdjCell> AdjList;
private:
    AdjList mAdjLists; //始点と終点の集合（複数）
    int mVertexCount;  //頂点の個数
    int mEdgeCount;    //辺の個数
```

リスト4　Graphクラス（コンストラクタとデストラクタ）

```
public:
    //コンストラクタ
    Graph(){ mVertexCount = mEdgeCount = 0; }
    ...（略）...
    //デストラクタ
    virtual ~Graph(){ clear(); }
    //内容の破棄
    void clear(){
        typename AdjList::iterator aItr;
        for(aItr = mAdjLists.begin();
                aItr != mAdjLists.end(); aItr++){
            VertexT* aVtx = aItr->startVertex;
            delete aVtx; //頂点の解放
        }
        mAdjLists.clear(); //隣接情報の破棄
        mVertexCount = mEdgeCount = 0;
    }
```

リスト5　Graphクラス（内部情報を返すメンバ関数）

```
    // 内部リストを返す
    AdjList& adjList() { return mAdjLists; }
    // iVertexと同値の始点を持つ「隣接情報（始点と終点の集合（一つ分））」
    //    へのイテレータを返す
    // 見つからない場合はadjList().end()を返す
    typename AdjList::iterator adjCell(const VertexT&
                                       iVertex) {
        typename AdjList::iterator aItr;
        for(aItr = mAdjLists.begin();
                aItr != mAdjLists.end(); aItr++){
            if(iVertex.equalVertex(aItr->startVertex)){
                return aItr;
            }
        }
        return aItr;
    }
    ...（略）...
    // 頂点の個数を返す
    int vertexCount() const { return mVertexCount; }
    // 辺の個数を返す
    int edgeCount() const { return mEdgeCount; }
```

リスト 6　Graphクラス(頂点や辺を登録するメンバ関数)

```
    // 頂点(iVertex)を登録する
    // 登録できたなら true,できなかったなら falseを返す
    bool insertVertex(const VertexT& iVertex) {
        //すでに頂点が登録されているかを確認する
        if(adjCell(iVertex) != mAdjLists.end())
            return false;
        //登録用の頂点を確保する
        VertexT* aVertex = new VertexT(iVertex);
        //頂点を登録する
        AdjCell aCell;
        aCell.startVertex = aVertex;
        mAdjLists.push_back(aCell);
        //頂点の個数を増やす
        ++mVertexCount;
        return true;
    }
    ...( 略) ...
    // 辺を登録する
    //   ( 始点 =iStartVertex,終点 =iEndVertex,辺 =iEdge)
    // 登録できたなら true,できなかったなら falseを返す
    bool insertEdge(const VertexT& iStartVertex,
                    const VertexT& iEndVertex,
                    const EdgeT&   iEdge = EdgeT()) {
        //始点を探す
        typename AdjList::iterator aStartP = adjCell(
                                                iStartVertex);
        if(aStartP == mAdjLists.end())
            return false;
        //終点がすでに登録されているかを探す
        EdgeList& aEList = aStartP->edges;
        typename EdgeList::iterator aEItr;
        for(aEItr = aEList.begin(); aEItr != aEList.end();
            aEItr++){
            //ある場合
            if(iEndVertex.equalVertex(aEItr->endVertex))
                return false; //登録できないので戻る
        }
        //終点を探す
        typename AdjList::iterator aEndP = adjCell(iEndVertex);
        if(aEndP == mAdjLists.end())
            return false;
        //登録用の辺を用意する
        EdgeT aEdge = iEdge;
        aEdge.endVertex = aEndP->startVertex;
        //辺を登録する
        aStartP->edges.push_back(aEdge);
        //辺の個数を増やす
        ++mEdgeCount;
        return true;
    }
    // 辺を登録する(点=iVertex1,iVertex2,辺=iEdge) 両方向対応
    void insertEdgeUD(const VertexT& iVertex1,
                      const VertexT& iVertex2,
                      const EdgeT& iEdge = EdgeT()) {
        insertEdge(iVertex1,iVertex2,iEdge);
        insertEdge(iVertex2,iVertex1,iEdge);
    }
    ...( 略) ...
};
```

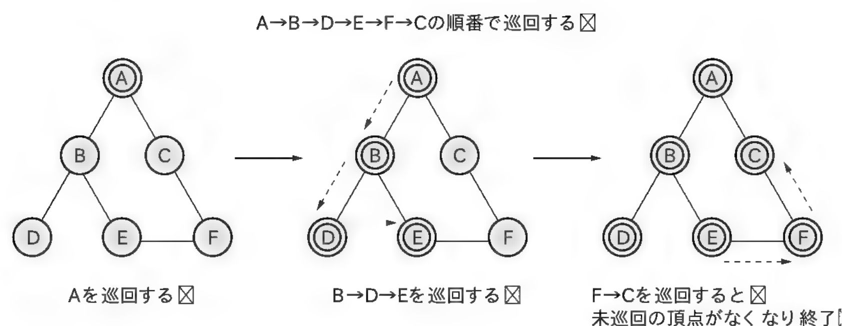


図 5　深さ優先探索

策すればよいのかはあえて説明せず，読者への課題としておきましょう．

最後に頂点や辺を登録するメンバ関数です(リスト 6)．説明するまでもありませんが，insertEdgeと insertEdgeUDの違いは有向グラフで使うか，無向グラフで使うかの違いだけです．

グラフの探索

グラフの頂点を巡回する処理は本連載の第 13回で出てきたバイナリ・ツリーの巡回と同様，頂点および頂点につながっている辺を巡回して再帰的に別の頂点をたどる方法だと考えられます．

● 深さ優先探索(depth-first search)

巡回した頂点が所持する 1個の辺につながる頂点を巡回し，さらにその頂点が所持する 1個の辺につながる頂点をどんどん再帰的に巡回する方法です(図 5)．巡回した先の頂点から別の未巡回の頂点が見つからない場合，一つ手前の頂点に戻り，そこで別の辺に対して同じ作業を繰り返します．そして巡回する辺も頂点もなくなった時点で探索が終了します．

● 幅優先探索(breadth-first search)

巡回した頂点が所持する辺につながる頂点をいったんキューに記録します(図 6)．さらにキューから取り出した頂点を巡回し，その頂点が所持する辺につながる頂点をキューに記録するという作業をキューが空っぽになるまで繰り返します．ある頂点を巡回したなら，すぐ隣の頂点を巡回し，それが終わったなら，さらにその隣の頂点を巡回する作業になり，ちょうど池に放り込まれた 1個の石によって波ができ，それがどんどん広がるような感じの探索を行います．

どちらの探索方法を採用するかは取り扱う課題によって変わってきます．どちらの方法を使っても同じ結果がえられる場合，筆者ならば幅優先探索を選びます．しかし，これには論理的な根拠があるわけでなく，単に個人的な嗜好にすぎないと思います．

● 探索クラスの実装

グラフ探索のメンバ関数をさきほどの Graphクラスに追加してもよいのですが，プログラムの見通しを良くするため，あえて別個のクラス(GraphSearch)として実装します．グラフの探索でやっかいなのは頂点を巡回するときに付随する作業が，取り扱う課題によって変わるため，その作業を直接コード中に記述してしまうと再利用しにくくなるという点です．そこで頂点を処理する作業をコールバックする手法で実装してみます．いろんなコールバックが考えられますが，単純化して以下のメ

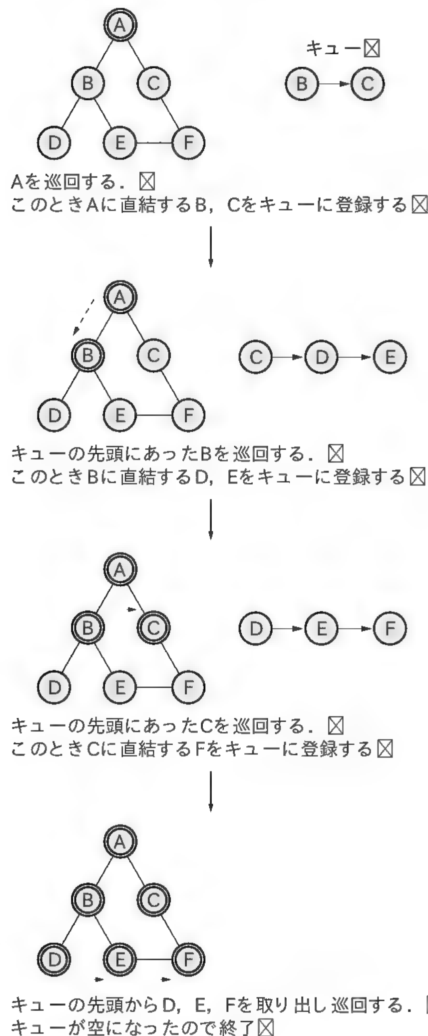


図6
幅優先探索

ンバ関数としてまとめます．

- simpleVertex ――全部の頂点を巡回して処理させる．初期化処理や終了化処理で必要になると思われるため
- beforeVertex ――幅優先探索でキューから頂点を取り出した段階で，その頂点を処理させる
- visitVertex ――通常はこのメンバ関数で巡回した頂点の処理をさせる
- afterVertex ――深さ優先探索で，その頂点につながっているすべての頂点の巡回が終了したことを知らせる

それぞれのメンバ関数をどう使えばいいかは後述します．コールバックのクラス（GraphWalker）はリスト7のように実装されます．通常はGraphWalkerを継承した独自のクラスを実装し，ここからコールバック・オブジェクトを発生させて，引き数に指定します．また，グラフ探索のクラスの先頭部分はリスト8のようになります．

コンストラクタでどのグラフを探索対象にするかを指定し，

リスト7　グラフ探索コールバックのクラス

```
template <class GraphT,class VertexT,class EdgeT>
class GraphWalker {
public:
    //simpleWalkで全頂点に対して呼ばれる
    //iVertex=頂点情報,ioMisc=汎用ポインタ
    virtual void simpleVertex(typename GraphT::AdjList::
                iterator iVertex,void* ioMisc) { /*(empty)*/ }
    //breadthFirstWalkで巡回キューから頂点を取り出したときに
    //  呼ばれる
    //iVertex=頂点情報,ioMisc=汎用ポインタ
    virtual void beforeVertex(typename GraphT::AdjList::
                iterator iVertex,void* ioMisc) { /*(empty)*/ }
    //breadthFirstWalkで巡回キューに頂点を登録するときに
    //  呼ばれる
    //depthFirstWalkで未巡回の頂点を発見したときに呼ばれる
    //iGraph=グラフ,iVertex=頂点情報
    virtual void visitVertex(typename GraphT::AdjList::
                iterator iVertex,void* ioMisc) { /*(empty)*/ }
    //depthFirstWalkですべての終点の巡回をおえて
    //  メンバ関数から抜ける直前に呼ばれる
    //iGraph=グラフ,iVertex=頂点情報
    virtual void afterVertex(typename GraphT::AdjList::
                iterator iVertex,void* ioMisc) { /*(empty)*/ }
};
```

リスト8　GraphSearchクラス（型宣言など）

```
//グラフ探索の関数群
template <class VertexT,class EdgeT>
class GraphSearch {
public:
    typedef Graph<VertexT,EdgeT> Graph_t;
private:
    Graph_t& mGraph; //探索対象のグラフ
    GraphSearch(); /* (empty) */
    //全頂点を未訪問にするコールバック・オブジェクト用
    class ClearVisitedWalker : public GraphWalker<
        Graph_t,VertexT,EdgeT> {
    public:
        virtual void simpleVertex(typename Graph_t::AdjList::
                     iterator iVertex,void* ioMisc) {
            VertexT* aVtx = iVertex->startVertex;
            aVtx->visited = false;
        }
    };
public:
    //コンストラクタ
    GraphSearch(Graph_t& iGraph) : mGraph(iGraph) {
                                          /*(empty)*/ }
    //全頂点を未訪問にする
    void clearVisited() {
        ClearVisitedWalker aCVW;
        simpleWalk(aCVW);
    }
    //グラフの全頂点を単純に巡回する
    //iGraphWalker=コールバック・オブジェクト,
    //ioMisc=汎用ポインタ
    void simpleWalk(GraphWalker<Graph_t,VertexT,EdgeT>&
                    iGraphWalker,void* ioMisc = NULL){
        typename Graph_t::AdjList& aAdjList = mGraph.adjList();
        typename Graph_t::AdjList::iterator aItr;
        for(aItr = aAdjList.begin(); aItr != aAdjList.end();
            aItr++)
            iGraphWalker.simpleVertex(aItr,ioMisc);
    }
```

探索前にclearVisitedメンバ関数で全頂点のvisited（訪問したかを判断するフラグ）をfalseにします．あるいはsimpleWalkメンバ関数を使って全頂点を初期化する処理でvisitedをfalseにするとよいでしょう．

simpleWalkはグラフの全頂点を単純に巡回するだけです．その際，全頂点が連結されている必要はなく，単純にグラフに

リスト 9 GraphSearchクラス(深さ優先探索)

```
//iStartVertex=巡回開始始点,iGraphWalker=コールバック
//  ・オブジェクト,ioMisc=汎用ポインタ
void depthFirstWalk(const VertexT& iStartVertex,
    GraphWalker<Graph_t,VertexT,EdgeT>& iGraphWalker,
    void* ioMisc = NULL){
    typedef typename Graph_t::AdjList::iterator
                                            DFWIterator;
    DFWIterator aItr; //巡回用イテレータ
    //巡回開始始点を探す
    aItr = mGraph.adjCell(iStartVertex);
    if(aItr == mGraph.adjList().end())
        return;
    //始点が巡回済みなら戻る
    if(aItr->startVertex->visited)
        return;
    //始点を巡回させる
    iGraphWalker.visitVertex(aItr,ioMisc);
    //始点を巡回済みにする
    aItr->startVertex->visited = true;
    //終点を巡回し,次の始点候補を探し,それを処理する
    typename Graph_t::EdgeList& aEList = aItr->edges;
    typename Graph_t::EdgeList::iterator aEItr;
    for(aEItr = aEList.begin(); aEItr != aEList.end();
        aEItr++){
        VertexT* aVertex = aEItr->endVertex; //終点をえる
        if(!aVertex->visited) //未巡回の終点であるなら
            //未巡回の終点を処理する
            depthFirstWalk(*aVertex,iGraphWalker,ioMisc);
    }
    //処理終了直前の頂点を知らせる
    iGraphWalker.afterVertex(aItr,ioMisc);
}
```

リスト 10 GraphSearchクラス(幅優先探索)

```
//iStartVertex=巡回開始始点,iGraphWalker=コールバック
//  ・オブジェクト,ioMisc=汎用ポインタ
void breadthFirstWalk(const VertexT& iStartVertex,
    GraphWalker<Graph_t,VertexT,EdgeT>& iGraphWalker,
    void* ioMisc = NULL){
    typedef typename Graph_t::AdjList::iterator
                                            BFWIterator;
    BFWIterator aItr; //巡回用イテレータ
    //巡回開始始点を探す
    aItr = mGraph.adjCell(iStartVertex);
    if(aItr == mGraph.adjList().end())
        return;
    //始点が巡回ずみなら戻る
    if(aItr->startVertex->visited)
        return;
    //始点を巡回させる
    iGraphWalker.visitVertex(aItr,ioMisc);
    //始点を巡回済みにする
    aItr->startVertex->visited = true;
    //巡回キューに始点を登録する
    std::list<BFWIterator> aVisitQueue; //巡回キュー
    aVisitQueue.push_back(aItr);
    //巡回キューに隣接情報(始点と終点の集合(一つ分))が
    //  ある限り繰り返す
    while(!aVisitQueue.empty()){
        //巡回キューの先頭にある頂点を取り出す
        aItr = aVisitQueue.front();
        iGraphWalker.beforeVertex(aItr,ioMisc);
        //巡回キューの先頭にある頂点から辺情報を取り出し,
        //  未巡回の終点を巡回キューに登録する
        typename Graph_t::EdgeList& aEList = aItr->edges;
        typename Graph_t::EdgeList::iterator aEItr;
        for(aEItr = aEList.begin(); aEItr != aEList.end();
            aEItr++){
            //終点をえる
            VertexT* aVertex = aEItr->endVertex;
            //未巡回の終点であるなら
            if(!aVertex->visited){
                //その終点の隣接情報へのイテレータをえる
                aItr = mGraph.adjCell(*aVertex);
                //終点を巡回させる
                iGraphWalker.visitVertex(aItr,ioMisc);
                //終点を巡回済みにする
                aVertex->visited = true;
                //巡回キューに終点を登録する
                aVisitQueue.push_back(aItr);
            }
        }
        //巡回キューの先頭を削除する
        aVisitQueue.pop_front();
    }
}
```

記録されている頂点を平等に訪問します．このときコールバック・オブジェクトのsimpleVertexメンバ関数が呼ばれるので，必要な処理はそのメンバ関数内に記述します．

深さ優先探索はリスト 9のようになります．

見てわかるとおり，一つの頂点を巡回するとき，コールバック(iGraphWalker.visitVertexを記述した箇所)が起きます．その頂点につながる別の頂点を探し，それが未巡回ならば再帰的に処理を行います．すべての頂点を巡回し終えると，再びコールバック(iGraphWalker.afterVertexを記述した箇所)が起きます．この二つのコールバックの違いを第13回のバイナリ・ツリーの巡回でたとえるなら，visitVertexはpreorder，すなわち「自分の頂点を巡回してから辺をたどっていく作業」であり，afterVertexはpostorder，すなわち「すべての辺をたどり終えた後で自分の頂点を巡回する作業」に相当します．

幅優先探索はリスト 10のようになります．

同じく，このメンバ関数でも一つの頂点を巡回するときにコールバック(iGraphWalker.visitVertexを記述した箇所)が起きます．その後，この頂点はキュー(aVisitQueue)に登録されますが，キューから取り出されるときにもコールバック(iGraphWalker.beforeVertexを記述した箇所)が起きます．後者の使い道としてはvisitVertexの対象になる頂点の一つ手前の頂点を記録する用途などが考えられます．

深さ優先探索の使用例

深さ優先探索の使用例として「トポロジカル・ソート(topological sort)」と呼ばれる処理があります．

たとえば複雑な部品を組み立てて製品を作ろうとするとき，ある部品を組み込むよりも前に別の部品を組み込んでおかないといけないというような制約があり，順番をまちがえると組み立てられないといった状況があるとします．たとえばノートPCを組み立てるとき，

(A1)基板にCPUとメモリを挿す

→(A2)CPUクーラを取り付ける

→(A3)基板とキーボード，HD，CD-ROMを結線する

リスト 11　深さ優先探索の使用例

```
struct SampleVertex : public Vertex {
    std::string name; //ノードの名前
    virtual bool equalVertex(const Vertex* iVertex) const {
        const SampleVertex* aVertex
                = dynamic_cast<const SampleVertex*>(iVertex);
        return (aVertex != NULL) && (aVertex->name == name);
    }
    SampleVertex(const char* iName) : name(iName) {
                                                /*(empty)*/ }
};

typedef Edge<SampleVertex> SampleEdge;
typedef Graph<SampleVertex,SampleEdge> SampleGraph;

//頂点と辺の登録
static void initSampleGraph(SampleGraph& iGraph)
{
    SampleVertex aA1("A1"),aA2("A2"),aA3("A3");
    ...(略)...
    iGraph.insertVertex(aA1);
    ...(略)...
    iGraph.insertEdge(aA1,aA2);
    ...(略)...
}

class CollectStartVertexWalker : public GraphWalker<
  SampleGraph,SampleVertex,SampleEdge> {
public:
    //このメンバ関数では，どこの終点にもなっていない頂点の
    //  検出を行う
    virtual void simpleVertex(SampleGraph::AdjList::
                         iterator iVertex,void* ioMisc) {
        SampleGraph::EdgeList& aEList = iVertex->edges;
        SampleGraph::EdgeList::iterator aItr;
        for(aItr = aEList.begin(); aItr != aEList.end();
            aItr++){
            SampleVertex* aEVertex = aItr->endVertex;
            //終点になっているのでフラグを立てる
            aEVertex->visited = true;
        }
    }
};

typedef std::list<SampleVertex*> SampleVertexList;

class TopologicalSortWalker : public GraphWalker<SampleGraph,
                                  SampleVertex,SampleEdge> {
public:
    //このメンバ関数ではどこの終点にもなっていない頂点の収集と
    //  visitedフラグの消去をする
    virtual void simpleVertex(SampleGraph::AdjList::iterator
                                  iVertex,void* ioMisc) {
        SampleVertex* aVertex = iVertex->startVertex;
        if(aVertex->visited){
            aVertex->visited = false;
        }else{
            //ioMiscにはSampleVertexListオブジェクト(頂点を
            //収集するリスト)へのポインタがあるので，その対策
            SampleVertexList* aSVL = static_cast<
                                  SampleVertexList*>(ioMisc);
            //頂点を収集する
            aSVL->push_back(aVertex);
        }
    }
    //このメンバ関数ではトポロジカル・ソートされた頂点の
    //  収集を行う
    virtual void afterVertex(SampleGraph::AdjList::iterator
                                  iVertex,void* ioMisc) {
        //ioMiscにはSampleVertexListオブジェクト(頂点を収集
        //  するリスト)へのポインタがあるので，その対策
        SampleVertexList* aSVL = static_cast<
                                  SampleVertexList*>(ioMisc);
        SampleVertex* aVertex = iVertex->startVertex;
        aSVL->push_front(aVertex);
    }
};

//デモ
static void demo()
{
    std::cout << "* topological sort demo start *¥n";
    SampleGraph aGraph;
    GraphSearch<SampleVertex,SampleEdge> aGraphSearch(aGraph);
    //頂点と辺の登録
    initSampleGraph(aGraph);
    //どこの終点にもなっていない頂点の検出
    //この処理で全頂点のvisitedはfalseになる
    aGraphSearch.clearVisited();
    CollectStartVertexWalker aCSVW;
    aGraphSearch.simpleWalk(aCSVW);
    //この処理の後でvisitedがfalseのままの頂点は
    //  どこの終点でもない
    //どこの終点にもなっていない頂点の収集&深さ優先探索の
    //  ための初期化
    //どこの終点にもなっていない頂点を収集するリスト
    SampleVertexList aStartVertexList;
    TopologicalSortWalker aTSW;
    aGraphSearch.simpleWalk(aTSW,&aStartVertexList);
    //どこの終点にもなっていない頂点が存在しないなら，全体が
    //閉路か，あるいは頂点が1つもないグラフなので処理できない
    int aSVLCount = aStartVertexList.size();
    std::cout << "start vertex count = " << aSVLCount
              << std::endl;
    if(aSVLCount == 0)
        return;
    //深さ優先探索によってトポロジカル・ソートを行う
    //トポロジカル・ソートの結果を入れるリスト
    SampleVertexList aSortedVertexList;
    //この処理で全頂点のvisitedはfalseになる
    aGraphSearch.clearVisited();
    SampleVertexList::iterator aSVLItr;
    for(aSVLItr = aStartVertexList.begin();
        aSVLItr != aStartVertexList.end(); aSVLItr++){
        SampleVertex* aVertex = *aSVLItr;
        std::cout << "research vertex = " << aVertex->name
                  << std::endl;
        aGraphSearch.depthFirstWalk(*aVertex,aTSW,
                                    &aSortedVertexList);
    }
    //処理結果の表示
    std::cout << "result = ";
    for(aSVLItr = aSortedVertexList.begin();
        aSVLItr != aSortedVertexList.end(); aSVLItr++) {
        SampleVertex* aVertex = *aSVLItr;
        std::cout << "[" << aVertex->name << "] ";
    }
    std::cout << std::endl;
    std::cout << "* topological sort demo end *¥n";
}
```

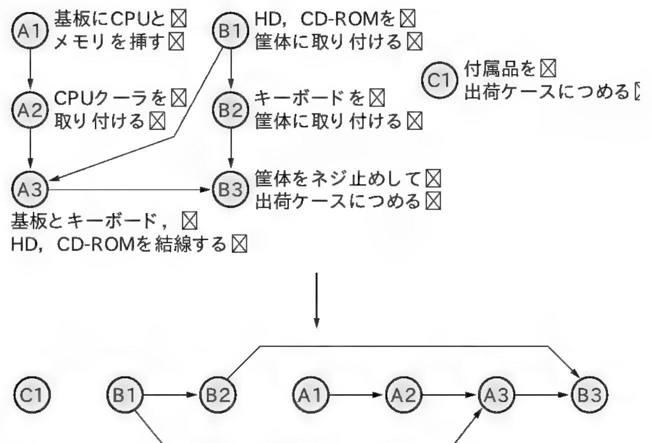


図7　トポロジカル・ソート

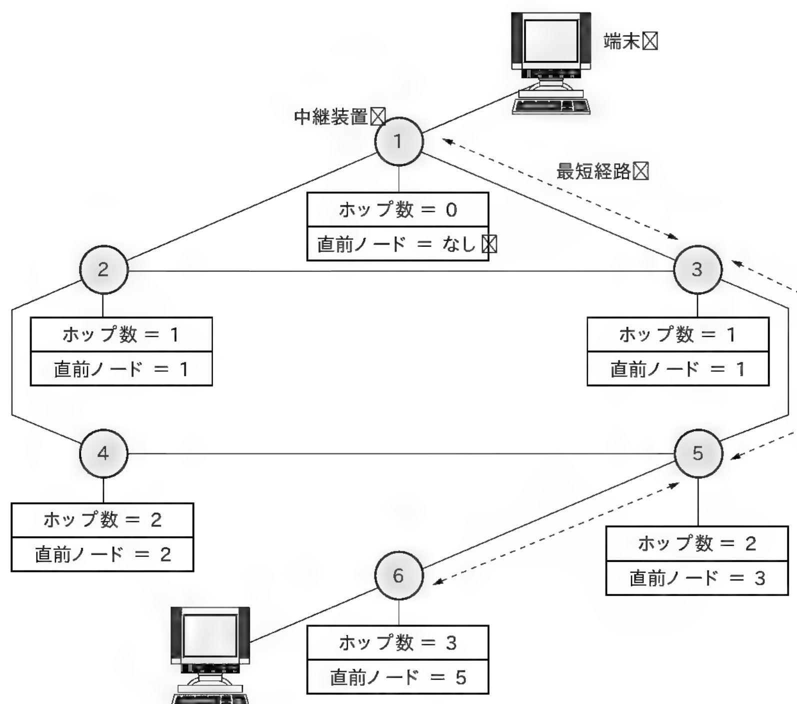


図8
ネットワーク上の通信

という基板中心の工程があり，同時に，

(B1)HD，CD-ROMを筐体に取り付ける

→(B2)キーボードを筐体に取り付ける

→(B3)筐体をネジ止めして出荷ケースにつめる

という筐体中心の工程があり，最後に，

(C1)付属品を出荷ケースにつめる

という工程があったとします．これらの工程を有向グラフで表現できた(図7，p.143)として，それぞれの工程を一人で行うなら，どの順番で行えば良いかが問題です．Aの工程もBの工程もそれぞれ連続しているほうが効率が良いのですが，途中で連続している流れを断ち切って違う工程に移る必要が出てきます．なるべく連続するように工程の順番を1本の流れに割り付ける作業がトポロジカル・ソートです．

トポロジカル・ソートの手順は以下のようになります．

1) 有向グラフからどの頂点の後にもならない頂点(始点)を探す．見つからない場合，頂点が一つもないか，全体が閉路なので処理を断念する
2) 1)で見つけた始点から深さ優先探索を行い，始点方向に舞い戻る時点(コールバックの`afterVertex`を使って判断できる)で頂点をスタックに登録する
3) すべての始点について2)の作業が終わった時点でスタックから頂点を取り出すと，それがトポロジカル・ソートされた結果となる

実際にトポロジカル・ソートを実装した例はリスト11(p.143)のようになります．

厳密にはトポロジカル・ソートの対象となる有向グラフに閉路があってはいけないのですが，閉路の検出は案外めんどうなので次回に説明します．ちなみに閉路がない有向グラフを「無閉路有向グラフ(directed acyclic graph，頭文字をとってDAGとも称する)」と称します．

幅優先探索の使用例

幅優先探索の例としてネットワーク上で中継装置を介して二つの端末が通信する状況を考えます(図8)．この状況をグラフにしたところ，頂点は中継装置を意味し，辺は中継装置どうしの接続状況を意味する無向グラフになったとします．このとき，なるべく中継が少なくなるルートを判断するのに幅優先探索が使えます．深さ優先探索と違い，幅優先探索は最初の頂点に直接つながる頂点を探索し，さらにそれらの頂点に直接つながる頂点への探索を波紋が広がるように行います．近所を優先して探索し，その後でじょじょに遠方に向かって探索の手をひろげるので中継回数を求める課題に適しています．

実際に探索を実装した例はリスト12のようになります．頂点には何段階中継しているかを判断するメンバ変数(`hopCount`)と，一つ手前の頂点を示すメンバ変数(`beforeNodeNo`)を用意しておきます．幅優先探索が終わるとすべての頂点の`hopCount`と`beforeNodeNo`がわかります．そして`beforeNodeNo`をたどれば，どの中継装置を経由すれば最短になるかを判断できます．

みやさか・でんと　miyadent@anet.ne.jp

リスト 12　幅優先探索の使用例

```
struct BFS_Vertex : public Vertex {
    int nodeNo;        //ノード番号
    int beforeNodeNo; //直前のノード番号
    int hopCount;      //ホップ数
    virtual bool equalVertex(const Vertex* iVertex) const {
        const BFS_Vertex* aVertex = dynamic_cast<
                            const BFS_Vertex*>(iVertex);
        return (aVertex != NULL) && (aVertex->nodeNo
                                        == nodeNo);
    }
};

typedef Edge<BFS_Vertex> BFS_Edge;
typedef Graph<BFS_Vertex,BFS_Edge> BFS_Graph;

static void dump_BFS_Graph(BFS_Graph& iGraph)
{
    std::cout << "* dump BFS_Graph start *¥n";
    std::cout << "  vertex count = " << iGraph.vertexCount()
    << " , edge count = " << iGraph.edgeCount() << std::endl;
    BFS_Graph::AdjList& aAdjList = iGraph.adjList();
    BFS_Graph::AdjList::iterator aItr;
    for(aItr = aAdjList.begin(); aItr != aAdjList.end();
        aItr++){
        BFS_Vertex* aVertex = aItr->startVertex;
        std::cout << "  v(nodeNo=" << aVertex->nodeNo;
        std::cout << ",beforeNodeNo="
                  << aVertex->beforeNodeNo;
        std::cout << ",hopCount=" << aVertex->hopCount;
        std::cout << ",visited=" << aVertex->visited << ") :";
        BFS_Graph::EdgeList& aEdgeList = aItr->edges;
        BFS_Graph::EdgeList::iterator aEItr;
        for(aEItr = aEdgeList.begin();
            aEItr != aEdgeList.end(); aEItr++)
            std::cout << " e(" << aEItr->endVertex->nodeNo
                      << ")";
        std::cout << std::endl;
    }
    std::cout << "* dump BFS_Graph end *¥n";
}

//頂点と辺の登録
static void init_BFS_Graph(BFS_Graph& iGraph)
{
    const int MinNodeNo = 1; //最小ノード番号
    const int MaxNodeNo = 6; //最大ノード番号
    //頂点の登録
    for(int aNodeNo = MinNodeNo; aNodeNo <= MaxNodeNo;
        aNodeNo++){
        BFS_Vertex aVertex;
        aVertex.nodeNo = aNodeNo;
        iGraph.insertVertex(aVertex);
    }
    //辺の登録
    struct EdgeReg_t {
        int from,to;
    };
    static EdgeReg_t aEdges[] = {
        {1,2},{1,3},{2,3},{2,4},{3,5},{4,5},{5,6},{-1,-1}
    };
    const EdgeReg_t *aEPtr = aEdges;
    for(bool aLoop = true; aLoop; ){
        if(aEPtr->from >= MinNodeNo){
            BFS_Vertex aFromV,aToV;
            aFromV.nodeNo = aEPtr->from;
            aToV.nodeNo = aEPtr->to;
            iGraph.insertEdgeUD(aFromV,aToV);
            ++aEPtr;
        }else{
            aLoop = false;
        }
    }
}

//コールバック
class BFS_Walker : public GraphWalker<BFS_Graph,BFS_Vertex,
                                      BFS_Edge> {
    BFS_Vertex* mLastVertex; //直前に巡回していた頂点
public:
    BFS_Walker() {
        mLastVertex = NULL;
    }
    virtual void simpleVertex(BFS_Graph::AdjList::
                          iterator iVertex,void* ioMisc) {
        BFS_Vertex* aVertex = iVertex->startVertex;
        aVertex->visited = false;    //この頂点を未巡回とする
        aVertex->beforeNodeNo = -1; //直前のノード番号
        aVertex->hopCount = 0;       //ホップ数
    }
    virtual void beforeVertex(BFS_Graph::AdjList::
                          iterator iVertex,void* ioMisc) {
        mLastVertex = iVertex->startVertex;
    }
    virtual void visitVertex(BFS_Graph::AdjList::
                          iterator iVertex,void* ioMisc) {
        if(mLastVertex != NULL){
            BFS_Vertex* aVertex = iVertex->startVertex;
            //直前のノード番号
            aVertex->beforeNodeNo = mLastVertex->nodeNo;
            //ホップ数
            aVertex->hopCount = mLastVertex->hopCount + 1;
        }
    }
};

//デモ
static void demo()
{
    BFS_Graph aGraph;
    GraphSearch<BFS_Vertex,BFS_Edge> aGraphSearch(aGraph);
    BFS_Walker aBFS_Walker;
    //頂点と辺の登録
    init_BFS_Graph(aGraph);
    //全頂点の情報をリセット
    aGraphSearch.simpleWalk(aBFS_Walker);
    //幅優先探索で処理
    BFS_Vertex aStartVertex;
    aStartVertex.nodeNo = 1;
    aGraphSearch.breadthFirstWalk(aStartVertex,aBFS_Walker);
    //処理結果の表示
    dump_BFS_Graph(aGraph);
}
```

x86CPUだけでもマスタしたい

開発技術者のためのアセンブラ入門

第㉘回　アセンブラを使いこなすための基礎知識と C言語からのアセンブラの使用方法（gas編：その1）

大貫 広幸

長かったこの連載も，今回を含め後2回となりました．連載最終の章となる今回と次回は，LinuxのC/C++言語のコンパイラgccから見たアセンブラの使用方法について説明します．

今回は，gccから呼び出されるサブルーチン（関数）をアセンブラgasで作成する方法について説明します．そして次回は，gccのインライン・アセンブラの機能とその使い方について説明する予定です．

リスト1　gasのロケーション・カウンタ（．，.org）

```
  1                    .data
  2 0000 0100                  .word   1
  3 0002 0200                  .word   2
  4                    lc1 = .
  5 0004 0300                  .word   3
  6 0006 0400                  .word   4
  7                    .text
  8 0000 90                    nop
  9 0001 00000000      . = .+4
 10 0005 90            lab1:   nop
 11 0006 90                    nop
 12 0007 90                    nop
 13 0008 8B1D0800              mov     .,%ebx
 13      0000
 14 000e 90                    nop
 15                    .data
 16 0008 0500                  .word   5
 17 000a 0600                  .word   6
 18 000c 00000000              .org    0x40
 18      00000000
 18      00000000
 18      00000000
 18      00000000
 19 0040 0700                  .word   7
 20 0042 0800                  .word   8
 21                    .text
 22 000f 90                    nop
 23                    lc2 = .
 24 0010 90                    nop
 25 0011 90                    nop
 26 0012 00000000              .org    0x84
 26      00000000
 26      00000000
 26      00000000
 26      00000000
 27 0084 90                    nop
 28 0085 8B1D8500              mov     .,%ebx
 28      0000
 29 008b 90                    nop

DEFINED SYMBOLS
            ap28_1.s:4       .data:00000004 lc1
            ap28_1.s:10      .text:00000005 lab1
            ap28_1.s:23      .text:00000010 lc2
```

gasではドット（．）で現在のロケーション・カウンタの値を取得できる☒

ドット（．）に対する演算も可能☒

.orgにより，.orgがあるセクションのロケーション・カウンタを指定値に変更できる☒

アセンブラgasを使いこなすために最低限必要な知識

最初にアセンブラgasでgccのサブルーチンを作成する場合，プログラミング上，最低これだけは覚えておいたほうがよいと思われることについて説明しておくことにします．

● アセンブラとアドレス

まずは，gasがアセンブル時に使うアドレスについてです．

（1）セクションとロケーション・カウンタ

MASMで「セグメント」と呼んでいたものを，gasでは「セクション」と呼びます．これは呼び方が異なるだけで，用途や使われ方は同じものといえます．そして，gasのロケーション・カウンタも，MASMと同じでセクションごとにあり，初期値はゼロになっています．

つまり，gasで使われる.dataや.textの各セクションは，すべて独自のロケーション・カウンタを持つわけです．gasの場合，現在のロケーション・カウンタの値はドット「．」により取得します．この「．」は特別なシンボルとしてgas自身が定義しているシンボルです．

使い方は，通常のラベルの参照と同じです．ただし，「．」は現在のロケーション・カウンタの値なので，参照するたびに，その値は異なります．

ロケーション・カウンタの値を変更するディレクティブとしては「.align」や「.org」などがあります．.alignは，連載第13回で説明しましたが，.orgについてはまだでした．

.orgは，二つオペランドを持ち，最初のオペランドで指定されたアドレスにロケーション・カウンタを強制的にセットします．そして，その指定されたアドレスまでの間を第2のオペランドで指定された値で埋めます．ただし，第2のオペランドを省略した場合はゼロで埋められます．

リスト1は，このロケーション・カウンタについての説明を実際のリストで示したものです．

（2）シンボルのアドレスの取得

以前から述べているように，gasのシンボルには型のような属性がありません．そのため，シンボルのアドレスを，命令のイミディエイトの値として欲しい場合は，シンボルの前に$を

付け記述することで取得します．

また，メモリの初期値としてシンボルのアドレスが欲しい場合は，シンボルをそのまま記述することで，メモリの初期値としてシンボルのアドレスが設定されます(リスト2)．

たとえば，シンボルがval100というデータがあった場合，「movl $val100, $ebx」とすることで，レジスタEBXにval100のアドレスがイミディエイト値として転送されます．また，「.long val100」とすることで，メモリ上にval100のアドレスが設定されます．

● ニモニック・サフィックス

これまでの説明では「gasのシンボルには型のような属性がないため，ニモニックの最後にデータ・サイズを表す文字を付ける」と説明してきました．このニモニックの最後に付けるデータ・サイズを表す文字のことを正式には「ニモニック・サフィックス」と呼びます．

gasでは，ニモニックにこのニモニック・サフィックスを付けるのを基本とします．しかし，別の要因でデータ・サイズがわかればニモニック・サフィックスを省略することができます．

このデータ・サイズがわかる別の要因としては，オペランドが一つ以上ある命令で，どれか一つのオペランドにレジスタが指定されている場合です．この場合，指定されたレジスタのサイズをデータ・サイズとすることができるからです(リスト3)．

たとえば「movw %ax, wVal1」は，レジスタAXの指定でデータ・サイズがワードと判断できるので「mov %ax, wVal1」と，ニモニック・サフィックスを省略して記述することができます．

逆のいいかたをするとオペランドからデータ・サイズがわからない命令では，必ずニモニック・サフィックスを付ける必要があるということです．たとえば「inc wVal1」は，オペランドからはデータ・サイズがわからないため，wVal1がワードだった場合は，必ずwのニモニック・サフィックスを付けて「incw wVal1」とする必要があります．

リスト2 gasのシンボルのアドレスの取得

```
 45 128e 3412          vaa:     .word    0x1234
 46 1290 0000                   .word    0
 47
 48 1292 A6000000               .long    lab1
 49 1296 8E120000               .long    vaa

 53 00a6 90            lab1:    nop
 54 00a7 BB8E120000             mov      $vaa,%ebx
 55 00ac BEA60000               mov      $lab1,%esi

DEFINED SYMBOLS
         ap28_2_4.s:45      .data:0000128e vaa
         ap28_2_4.s:53      .text:000000a6 lab1
```


リスト3 gasのニモニック・サフィックスの省略

```
  4 1284 3412          wVal:    .word    0x1234
  5 1286 78563412      lVal:    .long    0x12345678
  6 128a 00000000               .long    0

 12 0042 66A38412               movw     %ax,wVal
 12      0000
 13 0048 66A38412               mov      %ax,wVal
 13      0000
 14
 15 004e 03838612               addl     lVal(%ebx),%eax
 15      0000
 16 0054 13938712               adcl     lVal+1(%ebx),%edx
 16      0000
 17 005a 03838612               add      lVal(%ebx),%eax
 17      0000
 18 0060 13938712               adc      lVal+1(%ebx),%edx
 18      0000
 19
 20 0066 6641                   incw     %cx
 21 0068 6641                   inc      %cx
 22
 23 006a 66FF0584               incw     wVal
 23      120000
 24 0071 66FF4308               incw     8(%ebx)
 25
 26 0075 50                     pushl    %eax
 27 0076 50                     push     %eax
 28
 29 0077 66FF33                 pushw    (%ebx)
 30 007a FF33                   pushl    (%ebx)
 31 007c 66FF3584               pushw    wVal
 31      120000
 32 0083 FF358612               pushl    lVal
 32      0000
 33 0089 66683412               pushw    $0x1234
 34 008d 68785634               pushl    $0x12345678
 34      12
 35
 36 0092 58                     popl     %eax
 37 0093 58                     pop      %eax
 38 0094 668F03                 popw     (%ebx)
 39 0097 8F03                   popl     (%ebx)
 40 0099 668F0584               popw     wVal
 40      120000
 41 00a0 8F058612               popl     lVal
 41      0000
```


● ほかのモジュールへのシンボルの公開とほかのモジュールのシンボルの参照

.textで定義されているサブルーチンのエントリや.dataで定義されている初期値付きデータのシンボルを，ほかのモジュールに公開するには「.globlディレクティブ」を使用します．.globlは，公開したいシンボルをオペランドに記述することで，指定シンボルがほかのモジュールから参照可能となります．

たとえば，バイト・データのval1とラベルのfunc1という二つのシンボルを，ほかのモジュールから参照可能にするには，

```
    .globl    val1,    func1
```

と記述します．

初期値のないデータのシンボルは，「.commディレクティブ」で公開します．この.commディレクティブは連載第11回ですでに説明しました．

逆にほかのモジュールで定義されているシンボルを参照する場合は，gasでは特に何も必要ありません．そのままシンボル

リスト 4　gas の構造体

```
 58                      .data
 59 129a 0100            wDat01: .word   1
 60 129c 02000000                .long   2
 61
 62                              .struct    0
 63                      strcA_field1:   .space  4
 64                      strcA_field2:   .space  2
 65                      strcA_field3:   .space  3
 66                      strcA_End:
 67
 68                              .struct    0
 69                      strcB_field1:
 70                              .struct    strcB_field1+4
 71                      strcB_field2:
 72                              .struct    strcB_field2+2
 73                      strcB_field3:
 74                              .struct    strcB_field3+3
 75                      strcB_End:
 76
 77                      .data
 78 12a0 04000000                .long   4
 79 12a4 00000000        strcA:  .space  strcA_End
 79      00000000
 79      00
 80 12ad 0200                    .word   2
 81 12af 00000000        strcB:  .space  strcB_End
 81      00000000
 81      00
 82 12b8 01                      .byte   1
 83
 84                      .text
 85 00b1 A3A4120000              movl    %eax,strcA+strcA_field1
 86 00b6 66A3A8120000            movw    %ax,strcA+strcA_field2
 87 00bc A2AA120000              movb    %al,strcA+strcA_field3
 88 00c1 A2AB120000              movb    %al,strcA+strcA_field3+1
 89 00c6 A2AC120000              movb    %al,strcA+strcA_field3+2
 90
 91 00cb BBAF120000              movl    $strcB,%ebx
 92 00d0 894300                  movl    %eax,strcB_field1(%ebx)
 93 00d3 66894304                movw    %ax,strcB_field2(%ebx)
 94 00d7 884306                  movb    %al,strcB_field3(%ebx)
 95 00da 884307                  movb    %al,strcB_field3+1(%ebx)
 96 00dd 884308                  movb    %al,strcB_field3+2(%ebx)

DEFINED SYMBOLS
          ap28_2_4.s:59     .data:0000129a wDat01
                            *ABS*:00000000 strcA_field1
                            *ABS*:00000004 strcA_field2
                            *ABS*:00000006 strcA_field3
                            *ABS*:00000009 strcA_End
                            *ABS*:00000000 strcB_field1
                            *ABS*:00000004 strcB_field2
                            *ABS*:00000006 strcB_field3
                            *ABS*:00000009 strcB_End
          ap28_2_4.s:79     .data:000012a4 strcA
          ap28_2_4.s:81     .data:000012af strcB
```

gasでは.structによりアブソリュート・セクションが指定できる．このアブソリュート・セクション上のシンボルをオフセットとして使用することで構造体のフィールドにアクセスできる☒

アブソリュート・セクション上では初期値をもったデータは定義できない☒

上のstrcAは，このようにも記述できる☒

構造体を記憶するメモリの確保はこのようにする☒

構造体上のフィールドのアクセス☒

名を記述します．

アセンブル時には，このようなほかのモジュールで定義されているシンボルの参照は，未定義シンボルとなりますが，リンクによりほかのモジュールに定義があれば，この未定義は解決されます．しかし，リンクしても最終的に未定義のままのシンボルは，リンカが未定義シンボルのエラーとします．

gasでプログラミングする場合，この未定義シンボルの扱いがやっかいです．つまり，アセンブル時プログラム・ミスによる未定義シンボルがあった場合，最終的なリンクの段階にならないと，その未定義シンボルがわかりません．そのため，アセンブラのソースが複数ファイルある場合，リンカが出した未定義シンボルが，どのソース・ファイル上にあるのかを探すのがたいへんになるわけです．

この点，gasを使用する場合，注意が必要です．

● 構造体の定義とアクセス

gasには共用体はありませんが，構造体をアクセスするための機能があります．ただし，MASMなどの高級言語のような高度な機能ではなく，シンボルに格納するアドレスを絶対アドレスにするといっただけの機能です．そのため，フィールドに名前を付ける場合，その名前はそのモジュール内で一意である必要があります．つまり，一度使用したフィールド名は，モジュール内では再度フィールド名として使用することはできません．

構造体の開始は「.structディレクティブ」で指定します．.structディレクティブにはオペランドが一つあり，ロケーション・カウンタの開始絶対アドレスを指定します．構造体のアクセスは，たとえば構造体が入っているメモリのシンボルをstrcDatとすると，fid3という名前のフィールドをアクセスするとした場合，「strcDat+fid3」のように加算によってアクセスするアドレスを指定します．

実際のgasでの構造体の使い方はリスト4を参考にしてください．

gccのためのgasによるアセンブラ・サブルーチンの作成方法

ここでは，Linuxの32ビットC/C++コンパイラであるgcc(Ver egcs-2.91.66)，32ビット用のアセンブラgas(Ver 2.10.90)を使用して，C/C++言語のプログラムからアセンブラのサブルーチン(関数)を呼び出す方法について説明します．基本的なC/C++言語のプログラムからのアセンブラのサブルーチンの呼び出しは，前回述べたWindowsのVisual C++とMASMの場合とほぼ同じです．

● C/C++言語のプログラム側での準備

gccとgasでも，Visual C++とMASMのときと同じような作業を行います．

まず，C/C++言語のプログラム側で，呼び出すアセンブラのサブルーチンをプロトタイプ宣言し，C/C++コンパイラにこれから使用するアセンブラのサブルーチンを登録します．

アセンブラのサブルーチンの登録は，プロトタイプ宣言で通常のC言語の関数と同じように宣言します．このとき呼び出す側の言語がC++だった場合は，アセンブラのサブルーチンはC言語の関数として，

```
    extern "C" {
        // ここにアセンブラのサブルーチンを
```

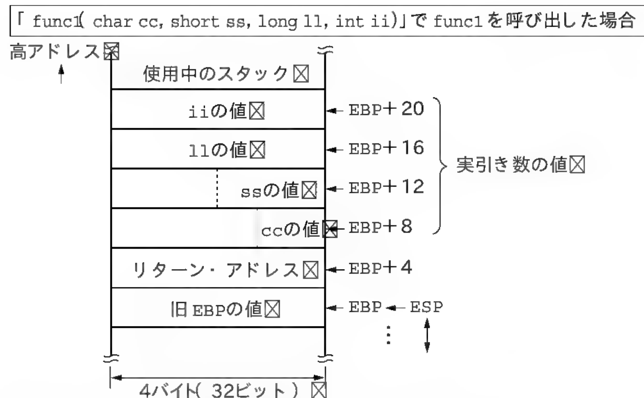


図1　gcc から C 言語の関数を呼び出したときの実引き数

```
        // C言語の関数として宣言する
    }
```

と宣言します．

アセンブラのプログラム側で定義されているデータ(変数)を，C/C++ 言語のプログラム側からアクセスするようなら，アセンブラのプログラム側で定義されているデータも C 言語の外部変数として extern で宣言しておきます．

gcc で使われる型とアセンブラ gas のデータの型は，表 1 のように対応しています．

● 呼び出し時の実引き数の受け渡し

gcc の場合も，実引き数はすべてスタックに PUSH された状態で渡されます．そして，整数やポインタの実引き数に対しては，型に関係なく，つねに値は long 型(32ビット，4バイト)に変換され，long 型の値としてスタックに PUSH されます．浮動小数点の値は，float 型の値なら 4バイト(32ビット)，double 型の値なら 8バイト(64ビット)でスタックに PUSH されます．

実引き数がスタックに PUSH される順番は，記述上から見て，右から左に向かって値が PUSH されます．たとえば，「func (aa, bb, cc)」の場合，まず cc が PUSH され，次に bb，そして最後に aa が PUSH されます．

アセンブラのサブルーチン側では，このスタック上の実引き数をレジスタ EBP を使いアクセスします．そのための処理をアセンブラのサブルーチンでは，最初に，

```
push    %ebp
movl    %esp,   %ebp
```

を実行します．先頭(左端)の実引き数は 8(%ebp)のアドレスにあります．

そして，サブルーチンからのリターンでは，

```
movl    %ebp,   %esp
pop     %ebp
ret
```

を実行します．

実引き数のスタックからの消去は，実引き数をスタックに PUSH した側，つまり呼び出した側で行います．

表1　gcc の変数の型とアセンブラ gas のデータ型の対応

gccのデータの型	gas の初期値付きデータ	バイト数
unsigned char	.byte	1
unsigned short	.word, .short	2
unsigned long	.long, .int	4
unsigned int	.long, .int	4
unsigned long long	.quad	8
char	.byte	1
short	.word, .short	2
long	.long, .int	4
int	.long, .int	4
long long	.quad	8
float	.float	4
double	.double	8
ポインタ	.long, .int	4

表2　アセンブラから gcc への戻り値の型と格納場所

戻り値	格納場所
unsigned char	32ビットにゼロ拡張後レジスタ EAX
unsigned short	32ビットにゼロ拡張後レジスタ EAX
unsigned long	レジスタ EAX
unsigned int	レジスタ EAX
unsigned long long	レジスタ EDX:EAX
char	32ビットに符号拡張後レジスタ EAX
short	32ビットに符号拡張後レジスタ EAX
long	レジスタ EAX
int	レジスタ EAX
long long	レジスタ EDX:EAX
float	FPU の ST(0)
double	FPU の ST(0)
ポインタ	レジスタ EAX

gcc の実引き数の受け渡しの状況を，図で表すと図 1 のようになります．

gcc の場合も，この呼び出し時の実引き数の受け渡しがアセンブラでプログラムできれば，アセンブラのプログラムから C 言語の関数を呼び出すことが可能です．

● アセンブラのサブルーチンからの戻り値

プロトタイプ宣言のとき戻り値を void とした場合，アセンブラのサブルーチンは戻り値を返す必要はありません．しかし，プロトタイプ宣言で戻り値の型を指定した場合は，その型でアセンブラのサブルーチンは戻り値を返す必要があります．

戻り値が整数値やポインタなら CPU のレジスタ EAX に戻り値を設定します．また，戻り値が浮動小数点の値なら FPU の ST(0)に戻り値を設定して戻ります(表 2)．

● アセンブラのサブルーチン側での注意

アセンブラのサブルーチン側では，CPU のレジスタ EAX，ECX，EDX の三つのレジスタは，退避することなく自由に使用できます．ほかの CPU レジスタを使う場合は，スタックにレジスタの値を退避し，呼んだルーチンに戻る前にレジスタの値を元の値に戻す必要があります．

図2　リスト5のプログラムのアセンブルおよびコンパイルと実行結果

```
[hiro@osdsrv LIST5]$ gcc -o cMain cMain.c asSub.s
[hiro@osdsrv LIST5]$ ./cMain
asFunc1 : 11 12 14 : 8
asFunc2 : 21 22 24 : 16
asFunc3 : 31 32 34 : 32
asFunc4 : 4.123450 -4.543210 : 1.234500
asFunc5 : 5.123450 -5.543210 : -9.876543
asFunc6 : [61234] : owari
[hiro@osdsrv LIST5]$ gcc -o cppMain cppMain.cxx asSub.s
[hiro@osdsrv LIST5]$ ./cppMain
asFunc1 : 11 12 14 : 8
asFunc2 : 21 22 24 : 16
asFunc3 : 31 32 34 : 32
asFunc4 : 4.123450 -4.543210 : 1.234500
asFunc5 : 5.123450 -5.543210 : -9.876543
asFunc6 : [61234] : owari
[hiro@osdsrv LIST5]$
```

アセンブルとコンパイルを別々に行ないたい場合は，次のようにする

```
as   -o  asSub.o  asSub.s
gcc  -o  cMain    cMain.c   asSub.o
```

```
as   -o  asSub.o   asSub.s
gcc  -o  cppMain   cppMain.cxx   asSub.o
```

FPUのレジスタは，レジスタ・スタックのみ使用できます．ただし，レジスタ・スタックは，呼んだルーチンに戻るとき，最初にそのアセンブラのルーチンが呼ばれた状態にする必要があります．ただし，浮動小数点の戻り値がある場合のみ`ST(0)`が戻り値に使われます．

フラグは，演算より変化するステータス・フラグ以外は，変化させないようにします．

● gccでは，C/C++言語側の関数や変数には，アンダスコア(_)は付加されない．

gccは，Visual C++と異なり，リンクに使用される関数や変数のシンボルには，アンダスコア(_)が頭に付きません．そのため，gcc側から呼び出されるアセンブラのサブルーチンやアクセスされるデータも，アセンブラ側のソースでは，アンダスコアを付加することなく，gcc側のソースで使用されている名前で記述することができます．

また，アセンブラ側から gcc側の関数や変数をアクセスする場合も，アセンブラ側のソースでは，アンダスコアを付加することなく，gcc側のソースで使用されている名前そのままを記述します．

● 実際のプログラム例

リスト5(pp.150-152)は，実際のC/C++言語のコンパイラgccとアセンブラgasでのプログラム例です．プログラムは，Linuxのコンソールで実行します．

このプログラムでは，C/C++言語のプログラムからアセンブラのサブルーチン呼び出しと，その逆のアセンブラのサブルーチンからC言語の関数呼び出し，そしてアセンブラのサブルーチンからのC/C++言語側の変数アクセス，C/C++言語のプログラムからアセンブラ側のデータのアクセスといったことを行っています．

図2は，このリスト5のプログラムのコンパイルと実行結果を示したものです．

＊　　＊

次回は，いよいよ連載の最終回として，gccのインライン・アセンブラの機能とその使い方について説明する予定です．

おおぬき・ひろゆき　大貫ソフトウェア設計事務所

リスト5　gccとgasの相互呼び出しのプログラム例

```
#ifndef __ASSUB_H
#define __ASSUB_H

#ifdef  __cplusplus
extern "C" {
#endif

/* アセンブラ上のデータの宣言 */
extern char   asVar_cc;
extern short  asVar_ss;
extern long   asVar_ll;
extern float  asVar_ff;
extern double asVar_dd;
extern char   *asPtr_ch;

/* アセンブラ上のサブルーチンの宣言 */
extern char   asFunc1(char cc,short ss,long ll);
extern short  asFunc2(char cc,short ss,long ll);
extern long   asFunc3(char cc,short ss,long ll);
extern float  asFunc4(float ff,double dd);
extern double asFunc5(float ff,double dd);
extern char   *asFunc6(unsigned short x);

#ifdef  __cplusplus
}
#endif

#endif /* __ASSUB_H */
```

アセンブラ側の変数やサブルーチンは，C++言語のときは「extern"C"」を指定し，C言語の変数や関数として宣言する

(a) C/C++言語が参照するアセンブラ側のデータとサブルーチンを宣言しているインクルード・ファイル(ファイル名は asSub.h)

リスト 5　gcc と gas の相互呼び出しのプログラム例（つづき）

```
/*
    C/C++ 言語とアセンブラ言語の相互呼び出し例
    << C 言語でのプログラム例 >>
*/

#include <stddef.h>
#include <stdio.h>

#include "asSub.h"

/* アセンブラからアクセスされる変数 */
char   ccVar_cc=0;
short  ccVar_ss=0;
long   ccVar_ll=0;
float  ccVar_ff=0.0;
double ccVar_dd=0.0;
char   *ccPtr_ch=NULL;

/* アセンブラから呼び出される関数 */
long ccFunc(unsigned short x,long y)
{
    long retVal;
    retVal = x + y;
    return retVal;
}

/* メイン関数 */
int main(void)
{
    char   rvVar_cc;
    short  rvVar_ss;
    long   rvVar_ll;
    float  rvVar_ff;
    double rvVar_dd;
    char   *rvPtr_ch;

    ccVar_cc = 8;
    ccVar_ss = 16;
    ccVar_ll = 32;
    ccVar_ff = 1.2345;
    ccVar_dd = -9.87654321;
    ccPtr_ch = "owari";

    rvVar_cc = asFunc1(11,12,14);
    printf("asFunc1 : %d %hd %ld : %d¥n",
                       asVar_cc,asVar_ss,asVar_ll,rvVar_cc);
    rvVar_ss = asFunc2(21,22,24);
    printf("asFunc2 : %d %hd %ld : %hd¥n",
                       asVar_cc,asVar_ss,asVar_ll,rvVar_ss);
    rvVar_ll = asFunc3(31,32,34);
    printf("asFunc3 : %d %hd %ld : %ld¥n",
                       asVar_cc,asVar_ss,asVar_ll,rvVar_ll);
    rvVar_ff = asFunc4(4.12345,-4.54321);
    printf("asFunc4 : %f %f : %f¥n",asVar_ff,asVar_dd,rvVar_ff);
    rvVar_dd = asFunc5(5.12345,-5.54321);
    printf("asFunc5 : %f %f : %f¥n",asVar_ff,asVar_dd,rvVar_dd);
    rvPtr_ch = asFunc6(1234);
    printf("asFunc6 : %s : %s¥n",asPtr_ch,rvPtr_ch);

    return 0;
}
```

(b) C 言語側のプログラム（ファイル名は cMain.c）

```
//
//    C/C++ 言語とアセンブラ言語の相互呼び出し例
//    << C++ 言語でのプログラム例 >>
//

#include <stddef.h>
#include <stdio.h>

#include "asSub.h"

// アセンブラからアクセスされる変数
extern "C" {
  char   ccVar_cc=0;
  short  ccVar_ss=0;
  long   ccVar_ll=0;
  float  ccVar_ff=0.0;
  double ccVar_dd=0.0;
  char   *ccPtr_ch=NULL;
}

// アセンブラから呼び出される関数
extern "C" long ccFunc(unsigned short x,long y)
{
    long retVal;
    retVal = x + y;
    return retVal;
}

// メイン関数
int main()
{
    char   rvVar_cc;
    short  rvVar_ss;
    long   rvVar_ll;
    float  rvVar_ff;
    double rvVar_dd;
    char   *rvPtr_ch;

    ccVar_cc = 8;
    ccVar_ss = 16;
    ccVar_ll = 32;
    ccVar_ff = 1.2345;
    ccVar_dd = -9.87654321;
    ccPtr_ch = "owari";

    rvVar_cc = asFunc1(11,12,14);
    printf("asFunc1 : %d %hd %ld : %d¥n",
                       asVar_cc,asVar_ss,asVar_ll,rvVar_cc);
    rvVar_ss = asFunc2(21,22,24);
    printf("asFunc2 : %d %hd %ld : %hd¥n",
                       asVar_cc,asVar_ss,asVar_ll,rvVar_ss);
    rvVar_ll = asFunc3(31,32,34);
    printf("asFunc3 : %d %hd %ld : %ld¥n",
                       asVar_cc,asVar_ss,asVar_ll,rvVar_ll);
    rvVar_ff = asFunc4(4.12345,-4.54321);
    printf("asFunc4 : %f %f : %f¥n",asVar_ff,asVar_dd,rvVar_ff);
    rvVar_dd = asFunc5(5.12345,-5.54321);
    printf("asFunc5 : %f %f : %f¥n",asVar_ff,asVar_dd,rvVar_dd);
    rvPtr_ch = asFunc6(1234);
    printf("asFunc6 : %s : %s¥n",asPtr_ch,rvPtr_ch);

    return 0;
}
```

(c) C++ 言語側のプログラム（ファイル名は cppMain.cxx）

リスト 5　gccと gas の相互呼び出しのプログラム例（つづき）

```
.data

# C/C++ 言語側からアクセスされるデータ
        .globl  asVar_cc,asVar_ss,asVar_ll
        .globl  asVar_ff,asVar_dd,asPtr_ch
asVar_cc:   .byte       0
asVar_ss:   .word       0
asVar_ll:   .long       0
asVar_ff:   .float      0.0
asVar_dd:   .double     0.0
asPtr_ch:   .long       0

# 整数値を文字列に sprintf 関数で変換するときに
# 使用するフォーマットとバッファ
strFmt: .asciz   "[%d]"
strBuf: .fill    80,1,0x20
        .byte    0

.text
#
# char asFunc1(char cc,short ss,long ll);
arg_cc = 8
arg_ss = 8+4
arg_ll = 8+4*2
        .globl  asFunc1
asFunc1:
        push    %ebp
        movl    %esp,%ebp
#
        movb    arg_cc(%ebp),%al
        movb    %al,asVar_cc
        movw    arg_ss(%ebp),%ax
        movw    %ax,asVar_ss
        movl    arg_ll(%ebp),%eax
        movl    %eax,asVar_ll
#
        movsbl  ccVar_cc,%eax  ← longより小さい戻り値は，符号付きの値は符号拡張，符号なしの値はゼロ拡張でEAXに設定する☒
#
        movl    %ebp,%esp
        pop     %ebp
        ret
#
# short asFunc2(char cc,short ss,long ll);
arg_cc = 8
arg_ss = 8+4
arg_ll = 8+4*2
        .globl  asFunc2
asFunc2:
        push    %ebp
        movl    %esp,%ebp
#
        movb    arg_cc(%ebp),%al
        movb    %al,asVar_cc
        movw    arg_ss(%ebp),%ax
        movw    %ax,asVar_ss
        movl    arg_ll(%ebp),%eax
        movl    %eax,asVar_ll
#
        movswl  ccVar_ss,%eax
#
        movl    %ebp,%esp
        pop     %ebp
        ret
#
# long asFunc3(char cc,short ss,long ll);
arg_cc = 8
arg_ss = 8+4
arg_ll = 8+4*2
        .globl  asFunc3
asFunc3:
        push    %ebp
        movl    %esp,%ebp
#
        movb    arg_cc(%ebp),%al
        movb    %al,asVar_cc
        movw    arg_ss(%ebp),%ax
        movw    %ax,asVar_ss
        movl    arg_ll(%ebp),%eax
        movl    %eax,asVar_ll
#
        movl    ccVar_ll,%eax
#
        movl    %ebp,%esp
        pop     %ebp
        ret
#
# float asFunc4(float ff,double dd);
arg_ff = 8
arg_dd = 8+4
        .globl  asFunc4
asFunc4:
        push    %ebp
        movl    %esp,%ebp
#
        flds    arg_ff(%ebp)
        fstps   asVar_ff
        fldl    arg_dd(%ebp)
        fstpl   asVar_dd
#
        flds    ccVar_ff
#
        movl    %ebp,%esp
        pop     %ebp
        ret
#
# double asFunc5(float ff,double dd);
arg_ff = 8
arg_dd = 8+4
        .globl  asFunc5
asFunc5:
        push    %ebp
        movl    %esp,%ebp
#
        flds    arg_ff(%ebp)
        fstps   asVar_ff
        fldl    arg_dd(%ebp)
        fstpl   asVar_dd
#
        fldl    ccVar_dd
#
        movl    %ebp,%esp
        pop     %ebp
        ret
#
# char *asFunc6(unsigned short x);
arg_x = 8
        .globl  asFunc6
asFunc6:
        push    %ebp
        movl    %esp,%ebp
#
        movl    $60000,%eax         ┐
        push    %eax                │ 実引き数をスタックにPUSHする場合は，longより小さい値は，符号付きなら符号拡張，符号なしならゼロ拡張し，longにしてからスタックにPUSHする☒
        movzwl  arg_x(%ebp),%eax    │
        push    %eax                ┘
        call    ccFunc  ← C言語の関数ccFuncの呼び出し☒
        addl    $4*2,%esp
#
        push    %eax
        pushl   $strFmt
        pushl   $strBuf
        call    sprintf  ← C言語のライブラリ関数sprintfの呼び出し☒
        addl    $4*3,%esp
        movl    $strBuf,%eax
        movl    %eax,asPtr_ch
#
        movl    ccPtr_ch,%eax
#
        movl    %ebp,%esp
        pop     %ebp
        ret
```

(d) アセンブラ側のプログラム（ファイル名は asSub.s）

TMS320C6713搭載DSPスタータ・キットを使った

C++による DSPオブジェクト指向プログラミング

第7回 FFTによるスペクトル・アナライザ

◆三上 直樹

今回は第5回目に作成した実信号用のFFTクラスを使って，スペクトル・アナライザのプログラムを作成します．このスペクトル・アナライザでは，得られるスペクトルを簡単に表示するため，計算されたスペクトルのデータをDSKボードに搭載されているCODECであるTLV320AIC23からアナログ量として出力し，オシロスコープに表示するようにします．

1 FFTによるスペクトル解析の方法

これから作成するプログラムでは，図1に示す手順でFFTを使ったスペクトル解析を行います．

● 窓掛け

最初に，標本化された信号に窓関数(window function)を乗算します．この処理を「窓掛け」といいます．今回のプログラムを作成する際には，方形窓，ハニング窓，ハミング窓，ブラックマン窓を使います．データの個数をLとすると，各窓関数を表す式は次のようになります．

▶ 方形(Rectangular)窓

$$w_R[n]=\begin{cases}1, & 0\leq n\leq L-1\\ 0, & \text{それ以外}\end{cases} \quad\cdots\cdots(1)$$

▶ ハニング(Hanning)窓

$$w_N[n]=\begin{cases}0.5-0.5\cos(2\pi n/L), & 0\leq n\leq L-1\\ 0, & \text{それ以外}\end{cases} \quad\cdots\cdots(2)$$

▶ ハミング(Hamming)窓

$$w_M[n]=\begin{cases}0.54-0.46\cos(2\pi n/L), & 0\leq n\leq L-1\\ 0, & \text{それ以外}\end{cases} \quad\cdots\cdots(3)$$

▶ ブラックマン(Blackman)窓

$$w_B[n]=\begin{cases}0.42-0.5\cos(2\pi n/L)+0.08\cos(4\pi n/L), & 0\leq n\leq L-1\\ 0, & \text{それ以外}\end{cases} \quad\cdots\cdots(4)$$

これらの窓関数を図2に示します．

● FFTによるDFTの計算とその絶対値

窓関数を乗算した信号のDFTの計算は，FFTを使って行います．その結果を$X[k]$とすると，この$X[k]$は一般に複素数になります．そこで，表示する際は$X[k]$の絶対値を表示することにします．また，信号については最初に特に断っていませんが，ここでは実数である信号，つまり実信号を考えています．その場合，使ったFFTの点数をMとすると，DFTの性質から次の式が成り立ちます．

$$X[k]=X[M-k]^*,\ k=(M/2)+1, (M/2)+2, \cdots, M-1 \quad\cdots\cdots(5)$$

なお，この式で，“*”の記号は複素共役を表します．

したがって，$X[k]$の絶対値を表示する場合，$k=0$の点を除くと，$k=M/2$を中心に左右対称になります．そのため，$k=0, 1, \cdots, M/2$に対応する$X[k]$の絶対値を表示します．

● dB化

スペクトル解析の結果を表示する場合，DFTの絶対値をそのまま表示すると，強度の弱い周波数成分がわかりにくくなり

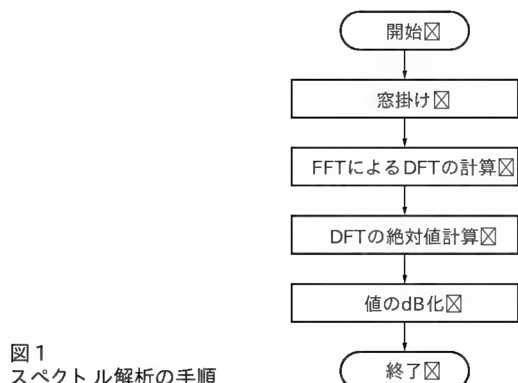


図1 スペクトル解析の手順

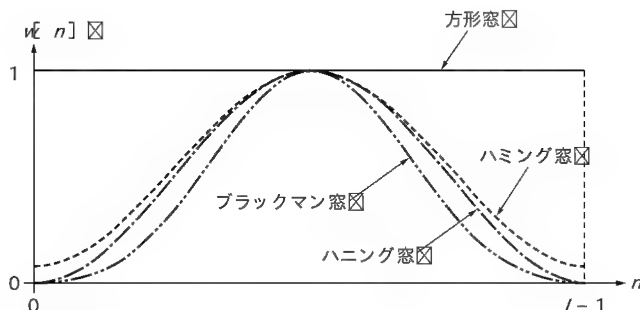


図2 窓関数

Column1 のこぎり波の補正

厳密に考える場合，図3の微分回路の出力にのこぎり波を出力するためには，どのような入力信号が必要になるのかを求める必要があります．しかし，ここでは話を簡単にするため，出力信号としてランプ(ramp)関数で表される波形が得られるような入力信号を考えます．そのように考えても，厳密に考えた場合と比べてそれほど大きな違いは出てきません．ランプ関数とは図A1(a)の左に示すようなものです．

微分回路にランプ関数で表される信号を入力したときの出力波形は図A1(a)の右に示すような波形になります．そのため，図A1(b)に示すように，微分回路にどのような波形を入力すれば，出力にランプ関数で表される波形が得られるのかを考えます．このような問題はラプラス変換(Laplace transform)を使うと簡単に計算できます．

微分回路の入出力の電圧に関する伝達関数$H(s)$，ランプ関数のラプラス変換$R(s)$はそれぞれ次のようになります．

微分回路の伝達関数

$$H(s)=\frac{s}{s+\frac{1}{CR}} \quad \cdots\cdots (A.1)$$

ランプ関数のラプラス変換

$$R(s)=\frac{A}{s^2} \quad A: 定数 \quad \cdots\cdots (A.2)$$

したがって，入力信号のラプラス変換を$X(s)$とすると，$X(s)$は次の式を満足すればよいことになります．

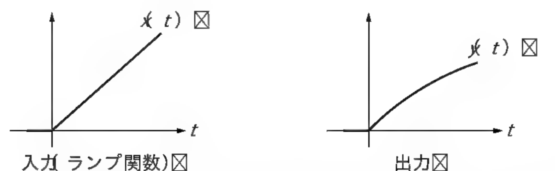
$$\frac{s}{s+\frac{1}{CR}}X(s)=\frac{A}{s^2} \quad \cdots\cdots (A.3)$$

この式から$X(s)$は次のようになります．

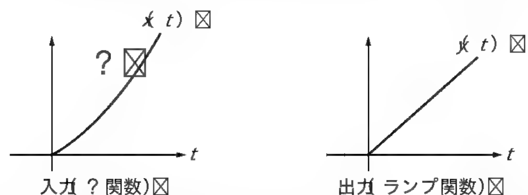
$$X(s)=\frac{A\left(s+\frac{1}{CR}\right)}{s^3} \quad \cdots\cdots (A.4)$$

この式の逆ラプラス変換を求めれば，必要な入力信号$x(t)$が求められ，次のようになります．

$$x(t)=At\left(1+\frac{1}{2CR}t\right) \quad \cdots\cdots (A.5)$$



(a) 微分回路にランプ関数を入力したときの出力信号の波形



(b) どのような関数を使うと，微分回路の出力がランプ関数になるか

図A1 のこぎり波の補正

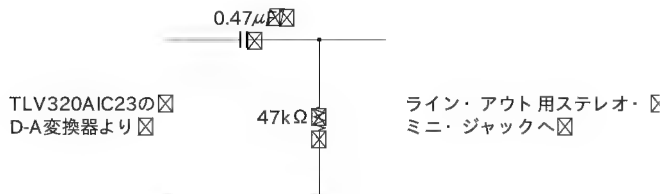


図3 TLV320AIC23のD-A変換器とライン・アウト用ステレオ・ミニ・ジャックの間のAC結合のようす

ます．そこで，それを防ぐためにDFTの絶対値をdB値に変換して表示します．つまり，表示する値は次のようになります．

$$20\log_{10}|X[k]|,\ k=0,\ 1,\ \cdots,\ M/2 \quad \cdots\cdots (6)$$

2 スペクトル解析結果の出力方法の検討

今回作るプログラムでは，スペクトル解析の結果をDSKボードに搭載されているCODECであるTLV320AIC23のD-A変換器から出力し，それをX-Yモードに設定したオシロスコープに表示します．つまり，X軸(水平軸)に対応するチャネルにはのこぎり波を，Y軸(垂直軸)に対応するチャネルにはスペクトルの値を加えます．このとき問題が一つあります．それは，D-A変換器の出力と出力のコネクタ(ステレオ・ミニ・ジャック)の間がAC結合されているということです．DSKボードの回路図を調べると，図3のような微分回路でAC結合されています．その影響で，D-A変換器から出力されるのこぎり波が歪んでしまうため，補正を行う必要があります．この補正の考え方は，コラム1に示します．その結果から，のこぎり波の時間とともに増加する部分は，次のような関数$f(t)$にすればよいことがわかります．

$$f(t)=At\left(1+\frac{1}{2CR}t\right) \quad \cdots\cdots (7)$$

A：定数，C：コンデンサの容量，R：抵抗値

プログラムを作る際は，この式を離散的時間に対応するものに変更します．つまり，標本化周波数をF_s，離散時間に対応する変数をn(n：整数)とすると，tをn/F_sに置き換えます．また，$a_0'=A/F_s$と置くと，式(7)は次のようになります．

$$f(n)=a_0'n(1+a_1n),\quad a_1=\left(\frac{1}{2CRF_s}\right) \quad \cdots\cdots (8)$$

リスト 1　のこぎり波の補正を検討するためのプログラム（SawtoothWave.cpp）

```
//------------------------------------------------------------
// のこぎり波発生のテスト
//      周期:        129
//      標本化周波数:      24 kHz
// このプログラムをビルドする際に，コンパイラ・オプションの
//    "General Debug Info" の項目を "No Debug" にしないように注意
//------------------------------------------------------------
#include "AIC23_Polling.hpp"

const int nFFT = 256;            // 使用するFFTの点数

// 表示のための定数
const int fMax = nFFT/2;         // 最大の周波数サンプル
const float C0 = 0.47e-6;        // C338 (0.47 μF)
const float R0 = 47e3;           // R342 (47 kΩ)
const float A1  = 1.2/(2.0*C0*R0*24000.0);   // fs = 24 kHzの場合
//const float A1  = 1.2/(2.0*C0*R0*48000.0); // fs = 48 kHzの場合
const int A00  = -32000/fMax;
const int A01  = -32000/(fMax*(1.0 + A1*fMax));

volatile int pos; // スライダの位置

// オブジェクトの宣言
AIC23_Polling codec(codec.fs24kHz);          // fs = 24 kHzの場合
//AIC23_Polling codec(codec.fs48kHz);        // fs = 48 kHzの場合

int main()
{
    short ch0, ch1;
    int nOut;

    pos  = 0;
    nOut = 0; // 周期カウンタ

    while (1)
    {
        codec.Read(ch0, ch1);    // ダミー読み出し

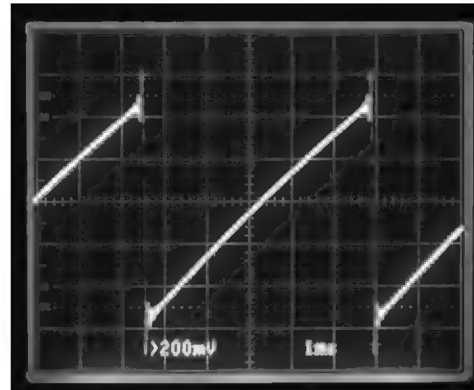
        if (pos == 0)
            ch0 = A00*nOut;
        else
            ch0 = A01*nOut*(1.0 + A1*nOut);

        codec.Write(ch0, ch1);   // 出力(CH0のみ有効)
        if (++nOut > fMax) nOut = 0;
    }
}
```

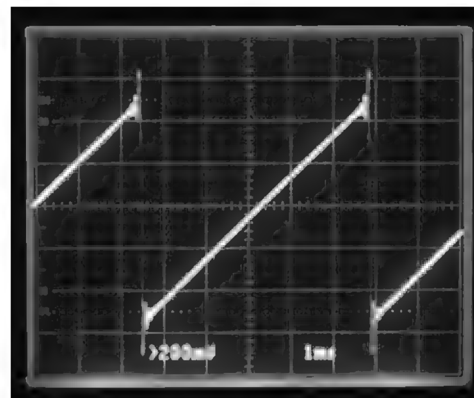
リスト 2　リスト 1（SawtoothWave.cpp）のプログラムで使うスライダを定義するGELファイルの内容

```
menuitem "Select01";

slider Select(0, 1, 1, 1, position)
{
    pos = position;
}
```



（a）補正なし



（b）補正あり

写真 1　のこぎり波の波形

この補正の効果を確認するために，のこぎり波を出力するプログラムを作りました．これをリスト 1（SawtoothWave.cpp）に示します．標本化周波数は24kHzに設定しています．なお，実際には図3の微分回路のコンデンサや抵抗器の値の誤差や，そのほかの要因も考えられるので，プログラムでは，a_1を式（8）の値ではなく式（9）の値を使い，この式のbは実験的に決めました．リスト 1ではこの値が1.2になっています．

$$a_1 = \left(\frac{b}{2CRF_s} \right) \cdots\cdots (9)$$

さらに，TLV320AIC23のD-A変換器に送るデータの極性には注意が必要です．つまり，D-A変換器から出力されるアナログ信号は，入力信号に対して極性が反転しています．そこで，D-A変換器には負の値を送る必要があるので，定数A00とA01は負の値になっています．なお，このプログラムでは，波形を出力している最中に補正の有無を切り替えるようにするため，Code Composer Studioが提供しているスライダを使っています．スライダを使うためのGELファイルの内容をリスト 2に示します．このスライダについては，コラム 2を参照してください．

写真1にこのプログラムを実行したときの波形を示します[注1]．写真 1（a）は補正がない場合で，のこぎり波の斜めの部分（左下から右上への部分）が直線ではなく上に凸である曲線になっているようすがわかります．一方，補正を行った写真（b）の場合は，のこぎり波の斜めの部分が直線になっています．

なお，この写真1を見るとわかるように，波形にはかなりのオーバシュート（overshoot）およびアンダシュート（undershoot）が見られます．これは，TLV320AIC23のD-A変換部にあるフィルタの過渡現象の影響で，基本的には取り除くことはできません[注2]．そこで，次項で作成するプログラムでは，X軸（水平軸）に対応するチャネルには図4（a）のようなのこぎり波では

注1：波形を観察するとき，オシロスコープの入力結合はDC結合にする必要がある．AC結合にすると，オシロスコープ内の微分回路の影響も現れる．

Column 2　スライダの使い方

Code Composer Studio(CCS)では，いろいろと便利な機能が提供されています．スライダもその一つです．スライダの機能を使うためには，まずGEL(General Extension Language)ファイルを作ります．その内容は，テキスト・データなので，CCS統合開発環境のエディタなどで作成することができます．ファイルの拡張子は“.gel”とします．

スライダの書式を図B1に示します．スライダのつまみはマウスで動かせますが，矢印キー(↑↓→←)やPage Upキー(Page Up)，Page Downキー(Page Down)でも動かすことができます．キーを1回押したときのつまみの移動量は，矢印キーを押した場合と，Page UpキーまたはPage Downキーを押した場合で，それぞれ別の値を設定することができます．

図B2にGELファイルの例を示します．1行目では，キーワードmenuitemを使って，メニュー・バーの“GEL”を選択すると現れる名前(ここでは“Volume Control”)の設定を行っています．スライダに関する記述はキーワードsliderから始まります．ここでは，スライダの目盛りの最小値を0，最大値を100に設定しています．つまみの移動量は，矢印キーに対して1に，Page UpキーとPage Downキーに対して10に設定しています．

スライダに関して図B3のように記述した場合，スライダのつまみの位置(position)がC/C++言語のソースで使っている変数(ここではlevel)に代入されます．

スライダの使い方は次のようになります．

1. [File | Load Gel...]を選択すると，GELファイルの選択画面が現れる．ここで目的のファイル(ここでは“volume.gel”)を選択し，[開く]ボタンをクリックする．または，目的のファイルがあるフォルダを開き，そこからProject Viewの“GEL files”へドラッグ＆ドロップする．ロードされたGELファイル名はProject Viewの“GEL files”の下に表示される(図B3)．ここに表示されたGELファイル名をダブルクリックすると，その内容が表示される．
2. 図B4に示すように，メニュー・バーの“GEL”を選択すると，GELファイルのmenuitemで定義された名前が現れるので，“Volume Control”を選択し，さらにスライダに対して付けられた名前(ここでは“Volume”)を選択すると，図B5に示すスライダが表示される．

参考までに，このスライダと組み合わせて動作するプログラムの例をリストB1に示します．このプログラムは，A-D変換器から入力された信号に，スライダで指定される0～100に対応する値(0.00～1.00，0.01刻みで指定)を乗算し，それをD-A変換器から出力するものです．

```
slider ParamDef(Min, Max, Inc, PageInc, ParamName)
{
  ここにスライダの動作を記述する ;
}
```

<説明>

ParamDef	スライダのタイトル・バーに表示される文字列
Min	最小値
Max	最大値
Inc	矢印キーが押されたときの移動量を指定
PageInc	PageUp，PageDownキーが押されたときの移動量を指定
ParamName	sliderの内部で使用されているパラメータの名前

図B1　キーワードsliderの書式

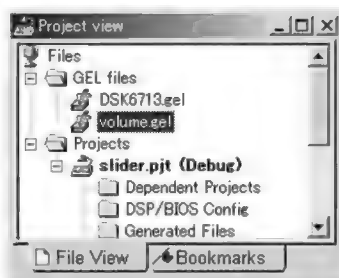
メニュー・バーの“GEL”を選択すると現れる項目の名前

```
menuitem "Volume Control";

slider Volume(0, 100, 1, 10, position)
{
  level = position;
}
```

スライダのつまみの位置

C/C++のソース・ファイルで定義されている変数名で，この変数につまみの位置に対応する数値が代入される



図B3　GELファイル“volume.gel”をロードしたようす



図B5　スライダ



図B4　“volume”を選択するとスライダ(図B5)が現れる

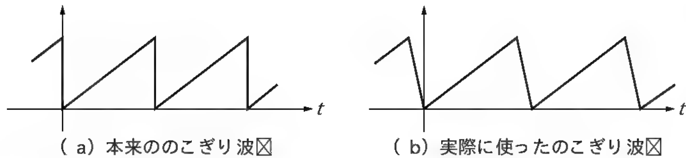
リストB1　スライダの使用例(VolumeCtrl.cpp)
A-D変換器からの入力信号のスライダで指定される0～100に対応する値(1/100刻みで0～1の範囲の数)を乗算し，D-A変換器へ出力するプログラム

```
//---------------------------
//   スライダの使用例
//---------------------------
#include "AIC23_Polling.hpp"

volatile int level;

int main()
{
    AIC23_Polling dsk;
    float ch[2], gain;

    while(1)
    {
        gain = level/100.0;
        dsk.Read(ch);
        for (int m=0; m<2; m++)
            ch[m] = ch[m]*gain;
        dsk.Write(ch);
    }
}
```



（a）本来ののこぎり波形　　（b）実際に使ったのこぎり波形

図4　のこぎり波の波形

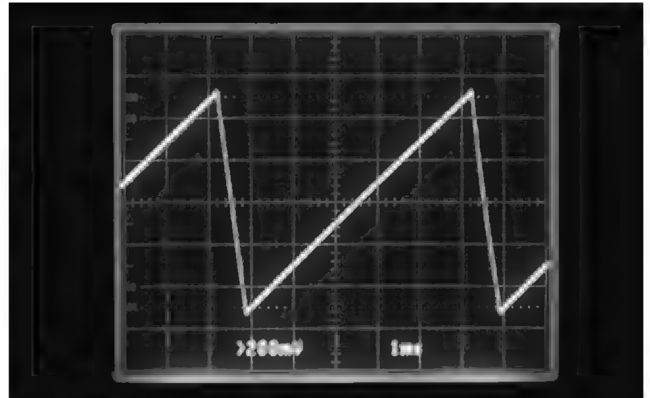


写真2　実際に使ったのこぎり波の波形

なく，図4（b）のようなのこぎり波を使うことにします．このプログラムは後にで示すリスト5の中の一部と同じため，特に示しませんが，そのときの波形は写真2のようになり，オーバシュートとアンダシュートがほとんど確認ができない程度にまで減少しているようすがわかります．

3 FFTによるスペクトル解析のプログラム（その1）

スペクトル解析のプログラムは，次の三つに分けて作成します．

1）窓掛け
2）スペクトルの計算
3）全体を統括するプログラム

● 窓掛け

ここでは4種類の窓関数を使いますが，方形窓のためのクラスRectWindowを基底クラスにします．次に，これを継承するクラスをTaperedWindowとし，残りの3種類の窓関数に共通な部分をまとめた抽象クラスとします．さらにクラスTaperedWindowを継承し，3種類の窓関数に対応するクラスHanninWindow，HamminWindow，BlackmenWindowを作成します．これらをまとめたものをリスト3（Windowing.cpp）に示します．図5には，これらのクラスの継承関係を示します．

▶ RectWindowクラス

リスト3のクラスは方形窓による窓掛けに対応するクラスで，ほかの窓関数に対応する派生クラスの基底クラスになります．

限定公開部で宣言されているArrayは，本連載の第5回目（本誌2004年6月号のp.163を参照）で作成した汎用1次元配列のためのクラスで，そのオブジェクトynは窓掛けを行った結果を格納するためのものです．Lは窓の幅，つまりデータ数（図2のL）に対応します．公開部ではコンストラクタ，デストラクタ，および窓掛けを実行する関数が宣言されています．

コンストラクタでは，メンバ初期設定の機能によりコンストラクタの引き数を使って，オブジェクトynと定数Lの初期化を行っています．それ以外には何も行いません．

デストラクタは仮想関数になっています．このクラスではデストラクタは何も行っていないので，必要ないと思うかもしれません．しかし，このクラスを継承する派生クラスが存在し，さらにその中にデストラクタが存在している場合に，そのデストラクタを呼び出すために必要になります．また，このクラスを継承する派生クラスにデストラクタが存在しない場合でも，デストラクタを持ったクラスのオブジェクトが派生クラスのメンバとして存在する場合も同様です．そのような場合にデストラクタが仮想関数として宣言されていなければ，メモリのリークを起こす場合があります[注3]．これについてはp.159のコラム3を参照してください．

窓掛けを実行するメンバ関数がExecute()です．方形窓では同じ重みを使うので，メンバ関数Execute()は何も処理を行わず，単にデータをコピーするだけです．残りの3種類の窓関数に対応する派生クラスでは，派生クラスで定義されるメンバ関数Execute()を使って窓掛けの処理を行うので，このメンバ関数は仮想関数（virtual function）にします．

▶ TaperedWindowクラス

RectWindowクラスの派生クラスとして，TaperedWindowクラスを作成します．このクラスは方形窓以外の窓関数に対応するクラスの親になるクラスです．そこでこのクラスは，それらのクラスに共通な部分をまとめて作ります．なお，このクラスは派生クラスを作ることを前提にしているので，抽象クラスにします．

限定公開部で宣言されているArrayのオブジェクトwnは，このクラスを継承する派生クラスのコンストラクタが計算する窓関数の値を格納するためのものです．Pi2Lは，派生クラスのコンストラクタの中で窓関数の値を計算するときに使う定数で，$2\pi/L$に対応する値です．

公開部ではコンストラクタ，デストラクタ，および窓掛けを実行する関数Execute()が宣言されています．

注2：オシロスコープに表示する際に，過渡現象が現れている区間はオシロスコープの輝度を下げることで，この影響を見えなくすることは可能である．しかし，そのためにはいろいろと細工が必要なので，今回は見送った．

注3：最初は筆者もこのことに気づかずにいた．しかし，同じプログラムをBorland社のC++ Builder6上でCodeGuard（メモリのリークなどを検出するツール）を有効にした状態で実行したところ，メモリのリークが発生することがわかったため，仮想デストラクタを追加した．

リスト 3　窓掛けを行うためのクラス

```
//-------------------------------------------------------------
// 窓掛けのためのクラス
//  方形窓, ハニング窓, ハミング窓, ブラックマン窓
//-------------------------------------------------------------
#ifndef MK_Windowing

#include "MyArray.hpp"
#include <cmath>
using namespace std;

//-------------------------------------------------------------
// 窓掛けのための基底クラス
class RectWindow
{
protected:
    Array<float>  yn;  // 窓掛けされたデータ
    const int      L;  // 窓の幅
public:
    RectWindow(const int n) : yn(n), L(n) {}
    virtual ~RectWindow() {}
    virtual Array<float> Execute(const float xn[])
    {
        for (int n=0; n<L; n++) yn[n] = xn[n];
        return yn;
    }
};
//-------------------------------------------------------------

//-------------------------------------------------------------
// RectWindowの派生クラスで, 他の窓関数用クラスの親になるクラス
class TaperedWindow : public RectWindow
{
protected:
    Array<float>  wn;       // 窓関数の値
    const float   Pi2L;     // 2π/L
public:
    TaperedWindow(const int n) : RectWindow(n), wn(n),
                                 Pi2L(6.283185/L) {}
    virtual ~TaperedWindow() = 0;
    Array<float> Execute(const float xn[])
    {
        for (int n=0; n<L; n++) yn[n] = wn[n]*xn[n];
        return yn;
    }
};

TaperedWindow::~TaperedWindow() {}

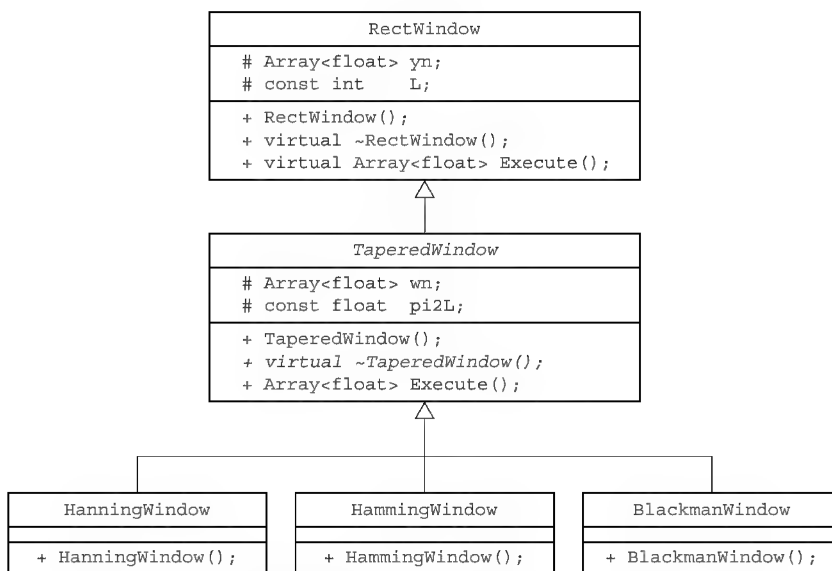
//-------------------------------------------------------------
// 以下の三つのクラスはTaperedWindowの派生クラス
//-------------------------------------------------------------

// ハニング窓
class HanningWindow : public TaperedWindow
{
public:
    HanningWindow(const int n) : TaperedWindow(n)
    {
        for (int k=0; k<L; k++)
            wn[k] = 0.5f - 0.5f*cosf(k*Pi2L);
    }
};

// ハミング窓
class HammingWindow : public TaperedWindow
{
public:
    HammingWindow(const int n) : TaperedWindow(n)
    {
        for (int k=0; k<L; k++)
            wn[k] = 0.54f - 0.46f*cosf(k*Pi2L);
    }
};

// ブラックマン窓
class BlackmanWindow : public TaperedWindow
{
public:
    BlackmanWindow(const int n) : TaperedWindow(n)
    {
        for (int k=0; k<L; k++)
            wn[k] = 0.42f - 0.5f*cosf(k*Pi2L)
                  + 0.08f*cosf(2.0f*k*Pi2L);
    }
};

#define MK_Windowing
#endif
```



+: 公開メンバ
#: 限定メンバ
(斜体は抽象クラスおよび純粋仮想関数クラスを表す)

図5　窓掛けに使用するクラスの継承関係

コンストラクタは，メンバ初期設定の機能を使って，基底クラス，オブジェクト wn，定数 Pi2L の初期化を行っています．それ以外には基底クラスと同様に何も行いません．

デストラクタは純粋仮想関数として宣言していますが，これは TaperedWindow クラスを抽象クラスにするためのものです．したがって，クラスの外部で記述されている定義を見るとわかるように，処理は何も行いません．

メンバ関数 Execute() は窓掛けを実行します．この関数は基底クラスのメンバ関数 Execute() をオーバライド(overriding)します．

▶ **HanningWindow クラス，HammingWindow クラス，BlackmanWindow クラス**

TaperedWindow クラスを継承するクラスとして，HanningWindow，HammingWindow，BlackmanWindow の三つのクラスを作成します．これらのクラスでは，コンストラクタで各クラス名に対応する窓関数の値を，式 2)，

Column 3 仮想デストラクタが必要な理由

基底クラスとそれを派生するクラスがあり，いずれもデストラクタを持っているときに，デストラクタを仮想関数にする必要が生じる場合があります．簡単な例として，次のような基底クラスBaseと派生クラスDerivedを考えます．

```
class Base  // 基底クラス
{
public:
    Base() { printf("Constructor of Base¥n"); }
    ~Base() { printf("Destructor of Base¥n"); }
};

class Derived : public Base  // 派生クラス
{
public:
    Derived() { printf(
                   "Constructor of Derived¥n"); }
    ~Derived() { printf(
                   "Destructor of Derived¥n"); }
};
```

これらのクラスを，以下のように基底クラスBaseのポインタptrが，実行時に派生クラスDerivedを指すようなプログラムで，デストラクタが正しく呼び出されるかを試してみます．

```
int main()
{
    Base *ptr = new Derived();
    delete ptr;
    return 0;
}
```

この実行結果は次のようになり，派生クラスDerivedのデストラクタが呼び出されていないことがわかります．

```
Constructor of Base
Constructor of Derived
Destructor of Base
```

そのため，派生クラスのデストラクタでメモリを解放するようなプログラムを作成した場合に，うまくメモリを解放できないことになります．

なぜこのようなことが起こるかということを次に説明します．

このプログラムでは，Baseクラスのポインタptrは，実行時にnew演算子によって生成されるDerivedクラスのオブジェクトを指すようになります．処理が終わり最後にptrを解放するとき，このポインタptrはBaseクラスのポインタとして宣言されているので，当然ですがBaseクラスのデストラクタは呼び出されます．しかし，プログラムのコンパイルの段階では，new演算子と対になって使われるdelete演算子はptrがDerivedクラスを指しているということはまだわかりません．そのため，Derivedクラスのデストラクタを呼び出すという実行コードを生成することはできないので，Derivedクラスのデストラクタは実行されません．

そこで，これを避けるために，Baseクラスのデストラクタを仮想関数にします．つまり，次のように~Base()の前にキーワードvirtualを追加します．

```
class Base  // 基底クラス
{
public:
    Base() { printf("Constructor of Base¥n"); }
    virtual ~Base() { printf(
                     "Destructor of Base¥n"); }
};
```

そうすると，delete演算子は，コンパイル時ではなく実行時にBaseクラスのポインタptrに，動的にバインディングされることになります．つまり，実行時に得られる情報に基づいてdelete演算子の動作は決められます．その結果，delete演算子はポインタptrが実行時にDerivedクラスのオブジェクトを指していることがわかるので，Derivedクラスのデストラクタを正しく呼び出せます．実行結果は次のようになり，派生クラスDerivedのデストラクタが呼び出されていることがわかります．

```
Constructor of Base
Constructor of Derived
Destructor of Derived
Destructor of Base
```

式(3)，式(4)を使って計算し，その値をTaperedWindowクラスのwnに格納します．

● スペクトルの計算

FFTを使ってスペクトルを計算するために使うクラスSpectrumAnalyzerのプログラムをリスト4(Spectrum.cpp)に示します．このクラスはFFTを使ってスペクトルを計算する以外に，データの格納と振幅スペクトルの値をdB値に変換して取り出す機能を持っています．

▶ 非公開データ・メンバ

非公開データ・メンバについてはリストのコメントを見るとわかると思いますが，一つだけ説明が必要でしょう．それはクラスRectWindowのオブジェクトを指すポインタの配列として宣言されているwPtrです．この宣言は実行時にポリモーフィズムを実現するために必要なものです．このポインタwPtrにはRectWindowのオブジェクトに対応するポインタだけでなく，RectWindowの派生クラスのオブジェクトに対応するポインタも代入することができます．

▶ コンストラクタ

コンストラクタでは，メンバ初期化の機能を使い，非公開データ・メンバの初期化を行います．この中には実数データのFFTを行うクラスRealFFTのオブジェクトRFFTの初期化も含まれています．

リスト 4　スペクトル解析器で使うクラス（Spectrum.cpp）

```
//---------------------------------------------------------------
// FFTによるスペクトル解析器で使用するクラス
//      （4種類の窓関数使用）
// © 三上直樹:  2004年
//---------------------------------------------------------------
#ifndef MK_SpectrumAnalyzer

#include "MyFFTReal.hpp"
#include "MyArray.hpp"
#include "Windowing.cpp"
#include <csl_dat.h>        // CSLのDATモジュール

//---------------------------------------------------------------
// SpectrumAnalyzerクラスの宣言部
class SpectrumAnalyzer
{
private:
    RealFFT        RFFT;     // 実データ用のFFTオブジェクト
    Array<Complex> Xk;       // FFTの出力用バッファ
    Array<float>   xn;       // FFTの入力用バッファ
    Array<float>   rxBuf;    // McBSP1からの受信データ用バッファ
    Array<float>   txBuf;    // McBSP1への送信データ用バッファ
    RectWindow     *wPtr[4]; // 窓掛け用クラスのポインタ
    const int      M;        // 使用するFFTの点数
    const int      M2p1;     // M/2+1
    const float    offset;   // スペクトルを表示する際の最小値を決める定数
    const float    scale;    // スペクトルを表示する際の倍率を決める定数
    const float    a1;       // スペクトルを表示する際の平滑化フィルタの係数
    const float    b0;       // スペクトルを表示する際の平滑化フィルタの係数
public:
    SpectrumAnalyzer(const int nFFT, const float ofs,
                     const float sc, const float a);
    ~SpectrumAnalyzer();
    void Execute(const int wtype);
    void rxPut(float xin, int n) { rxBuf[n] = xin; }
    inline short txGet(int n);
};

//---------------------------------------------------------------
// SpectrumAnalyzerクラスの定義部
// コンストラクタ
SpectrumAnalyzer::SpectrumAnalyzer(const int nFFT,
                    const float ofs,const float sc, const float a)
        : RFFT(nFFT), Xk(nFFT), xn(nFFT),
          rxBuf(nFFT), txBuf(nFFT/2+1, 0),
          M(nFFT), M2p1(nFFT/2+1),
          offset(ofs), scale(sc), a1(a), b0(1.0-a)
{
    DAT_open(DAT_CHAANY, DAT_PRI_LOW, 0);
    wPtr[0] = new RectWindow(M);
    wPtr[1] = new HanningWindow(M);
    wPtr[2] = new HammingWindow(M);
    wPtr[3] = new BlackmanWindow(M);
}

// デストラクタ
SpectrumAnalyzer::~SpectrumAnalyzer()
{
    DAT_close();
    for (int n=0; n<4; n++) delete wPtr[n];
}

// FFTによる，平滑化された振幅スペクトルの計算
void SpectrumAnalyzer::Execute(const int wtype)
{
    int xferID;

// 受信バッファのデータをFFTの入力バッファへ転送
    xferID = DAT_copy(rxBuf, xn, sizeof(float)*M);
    DAT_wait(xferID);

    xn = wPtr[wtype]->Execute(xn);    // 窓掛け
    RFFT.Execute(xn, Xk);             // FFTの実行
    for (int k=0; k<=M2p1; k++)       // 平滑化
        txBuf[k] = a1*txBuf[k] + b0*Norm(Xk[k]);
}

// 表示用に，周波数成分をdB値に変換して，取り出す
inline short SpectrumAnalyzer::txGet(int n)
{
    short yout;

    yout = 10.0f*log10f(txBuf[n]) - offset; // dB値に変換
    yout = (yout >= 0.0) ? -scale*yout : 0.0;

    return yout;
}

#define MK_SpectrumAnalyzer
#endif
```

コンストラクタの引き数は，次のようになっています．

- nFFT：使用するFFTの点数
- ofs　：スペクトル表示の際の最小値（dB単位）を決める定数
- sc　 ：スペクトル表示の際の，表示のスケールを決める定数．たとえば，表示のフルスケールを80dBに対応させる場合，32,768/80を与える
- a　　：スペクトル表示の際に，表示の変化の滑らかさを決める定数で0＜a＜1の範囲の数を与える．aが1に近いほど変化が滑らかになる

このコンストラクタで行う処理としては，まずEDMA（enhanced direct memory access）を使ってデータ転送を高速に行うために使うDATモジュール[注4]の初期化をDAT_open()により行います．次に，クラスRectWindowのオブジェクトのポインタに，new演算子を使って4種類の窓関数に対応するオブジェクトを動的に割り当てています．

▶ デストラクタ

デストラクタでは，DATモジュールをクローズし，コンストラクタで割り当てた窓関数クラスのオブジェクトを解放します．

▶ メンバ関数 **Execute()**

このメンバ関数は，FFTを利用してスペクトルの計算を行います．窓掛けの部分は，ポリモーフィズムを使って実現しています．

この関数では，最初にA-D変換されたデータが格納されている受信用バッファ（rxBuf）から，FFTの入力用バッファ（xn）へDATモジュールを使ってデータを高速に転送します．

次に，この関数の引き数に対応する窓関数による窓掛けを行います．この部分にはポリモーフィズムを利用しています．ここでは，どの窓関数を呼び出すかはコンパイル時には決まらず，実行時に初めて決まります．このような関数呼び出しは実行時バインディング[注5]（late binding）と呼ばれています．

窓掛けが行われたデータに対して，実数用のFFTでDFTの

注4：DATモジュールについては，本連載の第6回目（本誌2004年8月号p.166）を参照．
注5：実行時束縛，動的束縛（dynamic binding）などと呼ばれる場合もある．

リスト 5　FFTによるリアルタイム・スペクトル解析器のプログラム(FFT_Analyzer.cpp)

```
//-------------------------------------------------------------
// リアルタイム FFT解析器
//        FFTのデータ点数:    256
// このプログラムをビルドする際に,コンパイラ・オプションの
//        "General Debug Info"の項目を"No Debug"にしないように注意
//-------------------------------------------------------------
#include "AIC23_Intr.hpp"
#include "Spectrum.cpp"

const int nFFT = 256;        // FFTのデータ点数

// 表示のための定数
const int fMax = nFFT/2;     // 最大の周波数サンプル
const float C0 = 0.47e-6;    // C338 (0.47 μF)
const float R0 = 47e3;       // R342 (47 kΩ)
const float A1 = 1.2/(2.0*C0*R0*24000.0);
const int   A0 = -32000/(fMax*(1.0 + A1*fMax));

// 割り込みに関する設定のためのデータ
const AIC23_Intr::IntrConfig IntrCfTbl[] =
                  {  {IRQ_EVT_RINT1, 11},// McBSP1の受信割り込み
                     {0, 0} };            // 記述の終了記号

// オブジェクトの宣言
AIC23_Intr        codec(IntrCfTbl, codec.fs24kHz);
SpectrumAnalyzer  spa(nFFT, -30, 32768.0/80.0, 0.9);

volatile bool runFFT;        // true: FFT実行の許可
volatile int pos;            // スライダのつまみの位置

int main()
{
    short dummy[2];

    runFFT = false;          // FFT実行の禁止
    pos = 0;
    codec.Read(dummy);       // McBSP1のオーバラン・フラグのクリア

    IRQ_globalEnable();      // グローバル割り込み許可

    while (1)
        if (runFFT)
        {
            spa.Execute(pos);    // FFTによるスペクトル計算の実行
            runFFT = false;      // FFT実行の禁止
        }
}

// McBSP1の受信割り込み用の割り込みサービス・ルーチン
interrupt void AIC_RX_ISR()
{
    float ch0, ch1;
    short chshort[2];           // chshort[0]: x軸
                                // chshort[1]: y軸
    static int count  = 0;  // 入力データ数を数えるカウンタ
    static int nOut   = 0;  // 出力データのためのカウンタ
    static int updown = 1;  // nOutのインクリメント数またはデクリメント数

    codec.Read(ch0, ch1);   // 入力,CH1からの入力は使わない
    spa.rxPut(ch0, count);  // 受信データを受信バッファへ転送

    chshort[0] = A0*nOut*(1.0 + A1*nOut); // x軸
    if (updown > 0)
        chshort[1] = spa.txGet(nOut);      // スペクトルのdB値を取得
    else
        chshort[1] = 0;                    // 最低値

    if (nOut == 0) updown = 1;
    if ((nOut += updown) < 0) nOut = 0;
    if (nOut == fMax) updown = -8;

    codec.Write(chshort);   // 出力

    if (++count >= nFFT)
    {
        count = 0;
        runFFT = true;      // FFT実行の許可
    }
}
```

値を計算し,さらにDFTの絶対値の2乗を求めます.

最後に,スペクトルが時間とともに変動しているような信号のスペクトル値を表示した場合に,表示が滑らかに変化するように,1次のIIRフィルタをかけています.この計算は,現在のデータ・ブロックから計算された値を$x[n]$,現在表示する値を$y[n]$,一つ前に表示した値を$y[n-1]$とすると,次のようになります.

$$y[n]=ay[n-1]+(1-a)x[n] \quad \cdots\cdots\cdots\cdots (10)$$

表示の変化の滑らかさは,SpectrumAnalyzerのコンストラクタに与えられる第4引き数aにより決まり,aが1に近い値になるほど変化が滑らかになります.

▶ メンバ関数 **rxPut()**

この関数はA-D変換されたデータを受信用バッファ(rxBuf)に格納します.

▶ メンバ関数 **txGet()**

この関数は,メンバ関数Execute()の実行結果をdB値に対応する値に変換して取り出します.この値は,直接にCODECであるTLV320AIC23のD-A変換器に出力されることになります.ところで,本連載ではD-A変換器のビット幅を16ビットに設定しています.また,このD-A変換器からの出力は極性が反転された形になっています.したがって,D-A変換器には0～−32,768の範囲の数を送るようにしています.そのため,計算されたdB値にさらに−scaleという値を乗算しています.scaleの値は,SpectrumAnalyzerのコンストラクタの第3引き数scとして与えられています.

リスト 6　窓関数を切り替えるために使うスライダに対応するGELファイル(WindowSelect.gel)の内容

```
menuitem "WindowSelect";

slider Select(0, 3, 1, 3, position)
{
    pos = position;
}
```

● 全体をコントロールするプログラム

スペクトルの表示はオシロスコープを使います.オシロスコープはいわゆるX-Yモードに設定し,水平軸に対応するチャネル1にのこぎり波を入力し,垂直軸に対応するチャネル2にスペクトルの値を入力します.

リスト5(FFT_Analyzer.cpp)に,窓関数のためのクラスとスペクトル解析のためのクラスを使ったスペクトル・アナライザのプログラム全体を示します.このプログラムでは,コラム2で説明したスライダを使って,スペクトルを求める際の窓関数を切り替えます.そのスライダに対応するGELファイルの内容をリスト6(WindowSelect.gel)に示します.

標本化した信号の入力と表示するデータの出力の際には，McBSP1の受信割り込みを使います．したがって，全体はメイン関数と割り込み処理を行う関数`AIC_RX_IRS()`の二つに大きく分けられます．

▶ グローバル変数，およびオブジェクト，関数宣言など

使用するFFTの点数（`nFFT`）は256とします．`fMax`，`C0`，`R0`，`A1`，`A0`は結果を表示する際に使う定数です．

`IntrCfTbl[]`は割り込みを設定するための構造体の配列で，この設定によりMcBSP1の受信割り込みはCPU割り込みの11に割り当てられます．

クラス`AIC23_Intr`はTLV320AIC23を割り込みで使う際のクラスで，宣言されたオブジェクト`codec`は，標本化周波数が24kHzに設定されます．

クラス`SpectrumAnalyzer`のオブジェクト`spa`は，スペクトルを表示するときの基準値は−30dBに，垂直軸は80dBでフルスケールに，表示の変化を滑らかにするためのフィルタの定数は0.9にそれぞれ設定されます．

`volatile`付きで宣言されているグローバル変数は二つあります．`runFFT`はスペクトル計算の開始を許可/禁止するフラグです．`pos`はスライダのつまみに対応する値が代入される変数です．

▶ **`main()`**関数

メイン関数では，いくつかの初期設定を行った後，McBSP1がオーバラン・エラーを起こしている可能性があるので，一度`codec`のメンバ関数`Read()`でデータを読み出します．最後にグローバル割り込みを許可し，`while`文による無限ループに入ります．`while`文の中では，受信割り込み処理で`runFFT`が`true`に設定されたときに，`spa`のメンバ関数`Execute()`によるスペクトル計算の処理に移行します．

▶ 受信割り込み処理

この部分の処理では，最初にMcBSP1から読み出した入力信号を`spa`オブジェクトのバッファへ送ります．次にオシロスコープの水平軸に対応する入力端子に接続されているチャネルにのこぎり波を与えます．このとき，のこぎり波の立ち上がりの部分では，オシロスコープの垂直軸に対応する入力端子に接続されているチャネルにスペクトルの値を出力します．のこぎり波の立ち下がりの部分では表示する際の最小値を出力します．このようすがわかるように，写真3に通常の2チャネル表示を行った場合の波形を示します．上の波形がスペクトルに対応するもの，下の波形がのこぎり波です．

その後，のこぎり波を作るための処理を行い，2チャネル分の値をTMS320AIC23のD-A変換器へ送ります．

最後に，入力されたデータ数をカウントし，その値がFFTの点数である256に達した場合は，`runFFT`を`true`に設定し，スペクトル計算の処理に移行するための許可を与えます．

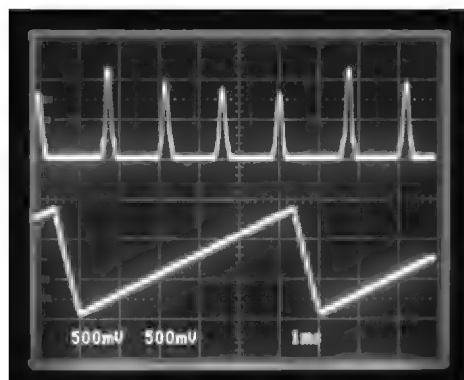


写真3　スペクトルを表示する際に水平軸および垂直軸に対応するチャネルに加える波形

● 実行結果

基本周波数1.6kHzの方形波に対する実行結果を写真4に示します．(a)は方形窓を使った場合で，(b)はハニング窓を使った場合です．

この結果を検討します．振幅1，基本周波数の方形波をフーリエ級数展開すると，次のようになります．

$$x(t)=\frac{4}{\pi}\left\{\sin\left(2\pi f_0 t\right)+\frac{1}{3}\sin\left(3\cdot 2\pi f_0 t\right)+\frac{1}{5}\sin\left(5\cdot 2\pi f_0 t\right)+\cdots\right\}$$

$$=\frac{4}{\pi}\sum_{n=1}^{\infty}\frac{1}{2n-1}\sin\left((2n-1)2\pi f_0 t\right)\quad\cdots\cdots(11)$$

この式から，基本周波数1.6kHzの方形波の場合，12kHzまでの周波数範囲で，その周波数成分は表1のようになります．

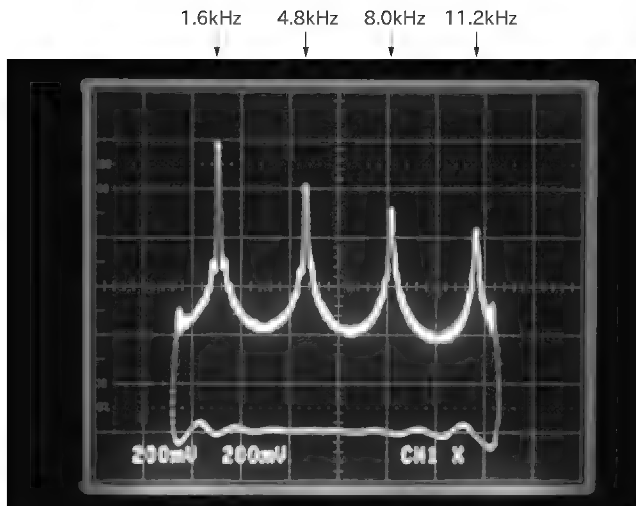
表1と写真4に示される結果を比較すると，スペクトルがほぼ正確に表示されていることがわかります．

また，使った窓関数による違いは，各窓関数のスペクトルにおけるサイド・ローブ（side lobe）の大きさの違いに現れています．方形窓を使った場合は，このサイド・ローブの影響が大きく現れています．一方，ハニング窓を使った場合は，サイド・ローブの影響が小さくなっています．詳しくは，文献（1）などを参照してください．

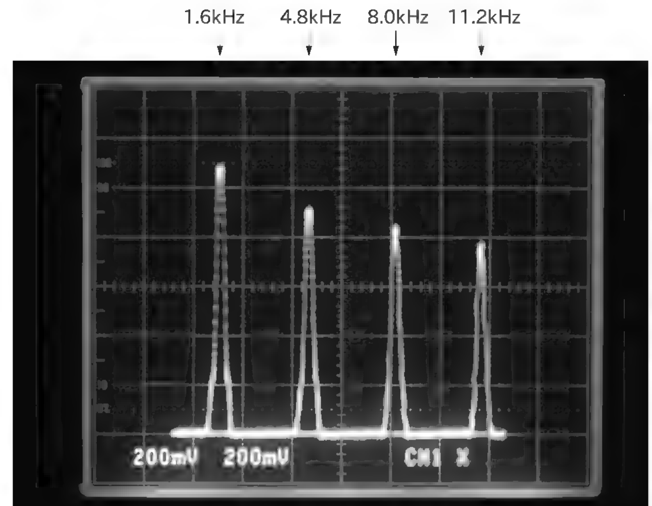
次に，エイリアシングの影響が現れた場合の実行結果を示します．基本周波数1.4kHzの方形波に対し，ハミング窓を使って実行した結果を写真5（a）に示します．本来は12.6kHz（1.4kHz×9）に対応する周波数成分が，エイリアシングの影響で11.4kHzの位置に現れています．次項ではこの点を改良したプログラムを示します．

表1　基本周波数1.6kHzの方形波の周波数成分

基本周波数に対する倍率	周波数（kHz）	基本波の振幅を1としたときの相対振幅
1	1.6	1
3	4.8	1/3
5	8.0	1/5
7	11.2	1/7

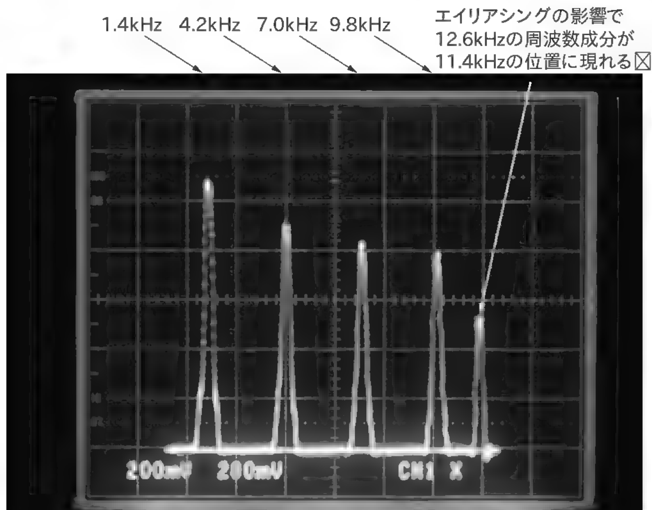


(a) 方形窓を使った場合

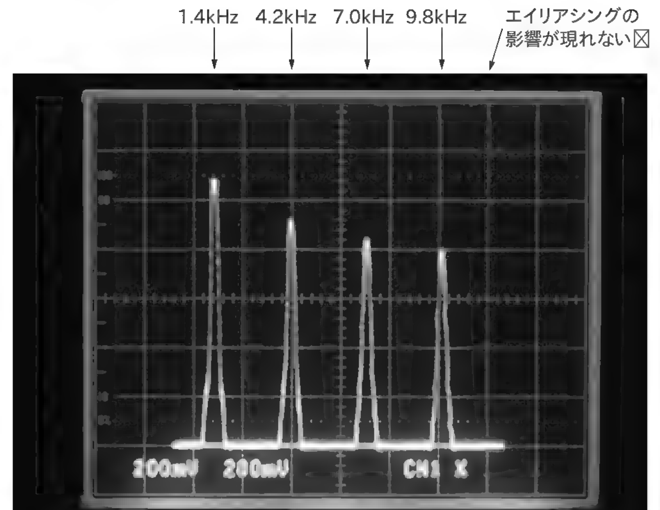


(b) ハニング窓を使った場合

写真4　スペクトルを表示しているようす
信号は 1,6kHz の方形波である



(a) エイリアシング防止処理を行わない場合



(b) エイリアシング防止処理を行った場合

写真5　エイリアシング防止処理の効果
信号は 1.4kHz の方形波，窓関数はハニング窓である

4 FFT によるスペクトル解析のプログラム(その2)

前項の実行結果のところで述べたようにリスト 5のプログラムはエイリアシングの影響が現れます．そこで，これを防止するため，マルチレート処理[2]を使ったプログラムを作成します．

● ダウン・サンプリング

リスト 5のプログラムでは標本化周波数を 24kHz に設定していましたが，TLV320AIC23に内蔵されているアンチエイリアシング·フィルタは，標本化周波数の 1/2の周波数，つまり 12kHz ではゲインが− 6dB になっており，エイリアシングの影響を十分に除くことはできません．そこで，標本化周波数を 48kHz に設定し，標本化された入力信号の 12～ 24kHz の成分をディジタル・フィルタで十分減衰させた後，標本化周波数を 24kHz として再び標本化するという操作を行います．このように，元の標本化周波数よりも低い周波数で標本化を行うことをダウン・サンプリングと呼びます．このようすを図 6に示します．図 6からわかるように，実際には低域通過フィルタを通ったデータを一つおきに間引く処理(decimation)を行うだけです．

通常のダウン・サンプリングでは，位相歪みのないフィルタをよく使います．そのため FIR フィルタが多く使われます．し

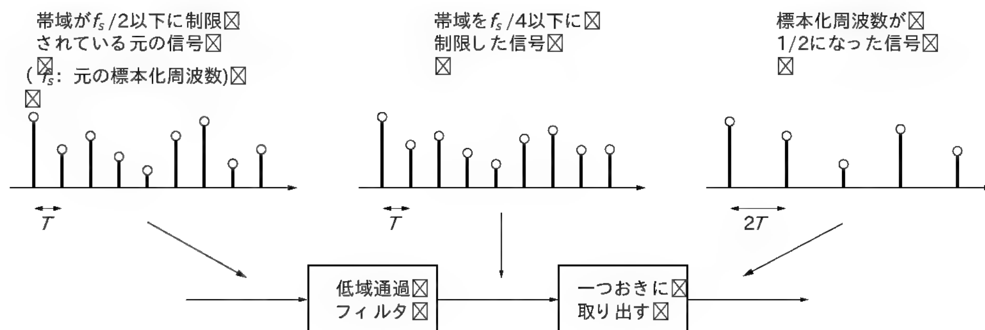


図6
標本化周波数を1/2に変換するようす

表2 ダウン・サンプリング用の低域通過IIRフィルタの設計時に与えたパラメータ

標本化周波数	48 kHz
通過域端の周波数	11.4 kHz
次数	9
通過域のリプル	0.5 dB
阻止域の減衰量	60 dB
振幅特性の形状	連立チェビシェフ特性

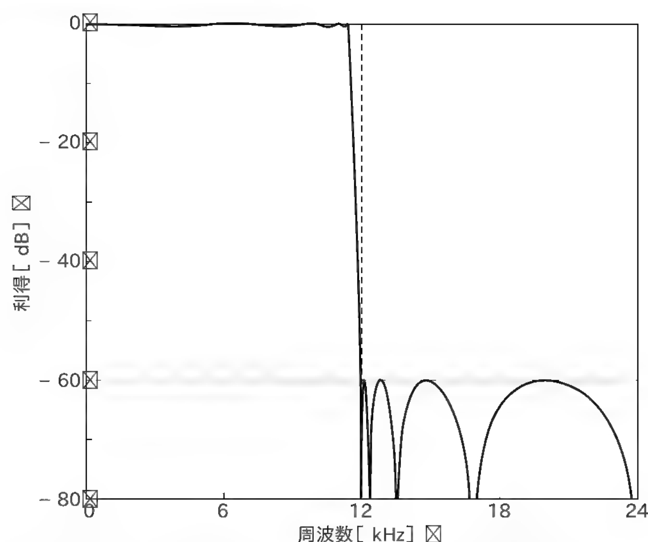


図7 標本化周波数を変換する際に使うフィルタの振幅特性

かし，ここでは振幅スペクトルだけを求めるので，位相ひずみがあっても問題にはなりません．そこで，ダウン・サンプリングに使うフィルタはIIRフィルタを使います．このフィルタは表2の仕様で，双一次z変換を使って設計しました[注6]．設計されたフィルタの振幅特性を図7に示します．このフィルタは12.0kHz以上の周波数成分を60dB以上減衰させせることができます．

● プログラム

ダウン・サンプリングで使う低域通過IIRフィルタのプログラムをリスト7(DirectIIR.cpp)に示します．このフィルタはテンプレート・クラスとして実現しました．フィルタの構造は直接形を使いました[注7]．

全体のプログラムをリスト8(FFT_AnalyzerDec.cpp)に示しますが，処理の大きな流れはダウン・サンプリングの部分を除くとリスト5と同じです．標本化周波数は48kHzに設定し，1/2にダウン・サンプリングした信号からスペクトルを計算します．

ダウン・サンプリングの処理は，受信割り込み処理AIC_RX_ISR()[注8]の中で行っており，網掛けした部分がその処理に対応します．まず，LPF.Execute()でIIRフィルタの処理を行います．次に，入力データ数をカウントしているcountが偶数の場合に，IIRフィルタの出力をspaオブジェクトのバッファへ送ります．これで1/2にダウン・サンプリングしたことになります．

受信割り込み処理の最後に，入力されたデータ数をカウントし，その値がFFTの点数の2倍である512に達した場合は，runFFTをtrueに設定し，スペクトル計算の処理に移行するための許可を与えます．

● 実行結果

p.163の写真5の場合と同じ信号，および同じ窓関数に対する実行結果を写真5(b)に示します．写真5(a)では，エイリアシングの影響で11.4kHzの位置に現れていた12.6kHzに対応する成分が，写真5(b)では現れなくなったことがわかります．

＊　＊

次回は最終回で，クラスを使ってIIRフィルタを実現した例について取り上げます．

注6：この設計法のプログラムは文献(3)の付録のCD-ROMに収録されている．
注7：IIRフィルタと，それをクラスとして実現する方法については次回に詳しく説明する．
注8：AIC_RX_ISR()での処理は，網掛け部分と最後のif文で条件を評価する部分(if(++count >= (nFFT << 1)))を除くと，リスト5と同じ．

参考文献
(1) 佐川，貴家："高速フーリエ変換とその応用"，第4章，昭晃堂，1993年．
(2) 貴家 仁志："マルチレート信号処理"，昭晃堂，1995年．
(3) 三上 直樹："C言語によるディジタル信号処理入門"，CQ出版社，2002年．

みかみ・なおき　職業能力開発総合大学校　情報工学科

リスト 7　リスト 8のプログラムで使用する直接型IIRフィルタ(DirectIIR.cpp)

```
//--------------------------------------------------
//    IIRフィルタ ( 直接形 II )
//--------------------------------------------------

template <int nOrd> class Direct2
{
private:
    float un[nOrd+1];
    const float *const ak, *const bk;
public:
    Direct2(const float am[], const float bm[])
            : ak(am), bk(bm)
    { for (int k=0; k<=nOrd; k++) un[k] = 0.0; }
    float Execute(const float xin);
};

template <int nOrd> float Direct2<nOrd>::Execute(
                                      const float xin)
{
    float yn, utmp;

    utmp = xin;
    for (int m=0; m<nOrd; m++) utmp = utmp + ak[m]*un[m+1];
    un[0] = utmp;
    yn = 0.0;
    for (int m=0; m<=nOrd; m++) yn = yn + bk[m]*un[m];
    for (int m=nOrd; m>0; m--) un[m] = un[m-1];

    return yn;
}
```

リスト 8　FFTによるマルチレート処理を使ったリアルタイム・スペクトル解析器のプログラム(FFT_AnalyzerMR.cpp)
網掛けした部分は，ダウン・サンプリングの処理に対応する

```
//--------------------------------------------------------------
// マルチレート処理を使ったリアルタイム FFT解析器
//        FFTのデータ点数:    256
// データの間引きに使ったIIR低域通過フィルタの設計パラメータ
//       type  楕円フィルタ
//       sampling frequency       48.0 kHz
//       passband edge frequency  11.4 kHz
//       order                    9
//       deviation in passband    0.5 dB
//       attenuation in stopband  60.0 dB
// このプログラムをビルドする際に，コンパイラ・オプションの
//        “General Debug Info”の項目を“No Debug”にしないように注意
//--------------------------------------------------------------
#include "AIC23_Intr.hpp"
#include "DirectIIR.cpp"
#include "Spectrum.cpp"

const int nFFT  = 256;      // FFTのデータ点数

// 表示のための定数
const int fMax = nFFT/2;    // 最大の周波数サンプル
const float C0 = 0.47e-6;   // C338 (0.47 μF)
const float R0 = 47e3;      // R342 (47 kΩ)
const float A1 = 1.2/(2.0*C0*R0*48000.0);
const int   A0 = -32000/(fMax*(1.0 + A1*fMax));

// 間引き用フィルタの定数
const int nOrdFilter = 9;   // 間引き用フィルタの次数
const float am[nOrdFilter] =
    {    2.1578163348, -4.8734858927,   6.3536124395,
        -7.3929472290,  6.2158964418,  -4.3277254340,
         2.2212361015, -0.8175472834,   0.1842101493};
const float bm[nOrdFilter+1] =
    {    0.02016445286, 0.05362744164, 0.12604956008,
         0.19368784956, 0.24593788198, 0.24593788198,
         0.19368784956, 0.12604956008, 0.05362744164,
                                       0.02016445286};

// 割り込みに関する設定のためのデータ
const AIC23_Intr::IntrConfig IntrCfTbl[] =
                     {  {IRQ_EVT_RINT1, 11}, // McBSP1の受信割り込み
                        {0, 0} };            // 記述の終了記号

// オブジェクトの宣言
AIC23_Intr   codec(IntrCfTbl);
SpectrumAnalyzer  spa(nFFT, -30, 32768.0/80.0, 0.9);

Direct2<nOrdFilter> LPF(am, bm);

volatile bool runFFT;  // true: FFT実行の許可
volatile int pos;      // スライダのつまみの位置

int main()
{
    ( リスト5 のmain関数と同じなので省略)
}

// McBSP1の受信割り込み用の割り込みサービス・ルーチン
interrupt void AIC_RX_ISR()
{
    float ch0, ch1, yn;
    short chshort[2];  // chshort[0]: x軸
    // chshort[1]: y軸
    static int count  = 0;      // 入力データ数を数えるカウンタ
    static int nOut   = 0;      // 出力データのためのカウンタ
    static int updown = 1;
                                // nOutのインクリメント数またはデクリメント数

    codec.Read(ch0, ch1);       // 入力，CH1からの入力は使わない

    yn = LPF.Execute(ch0);      // 低域通過フィルタの実行
    if ((count & 0x01) == 0)    // 受信データを受信バッファへ転送
        spa.rxPut(yn, count>>1);

    chshort[0] = A0*nOut*(1.0 + A1*nOut); // x軸
    if (updown > 0)
        chshort[1] = spa.txGet(nOut);     // スペクトルのdB値を取得
    else
        chshort[1] = 0;                   // 最低値

    if (nOut == 0) updown = 1;
    if ((nOut += updown) < 0) nOut = 0;
    if (nOut == fMax) updown = -8;

    codec.Write(chshort);       // 出力

    if (++count >= (nFFT<<1))
    {
        count = 0;
        runFFT = true;          // FFT実行の許可
    }
}
```

プログラミング入門シリーズ　好評発売中

C言語によるディジタル信号処理入門
Code Composer Studioを使ったDSPプログラミング

三上 直樹 著
B5変型判 296ページ　CD-ROM付き
定価2,940円(税込)
ISBN4-7898-3697-5

CQ出版社　〒170-8461 東京都豊島区巣鴨1-14-2　販売部 TEL.03-5395-2141　振替 00100-7-10665

組み込みプログラミング・ノウハウ入門(第18回)

イベント・ドリブンとフェイル・ソフト

――故障しても，かろうじて動き続けるシステム

藤倉 俊幸

前回は，RTOSなしでも利用できる時間駆動型の動的アーキテクチャであるCyclic Executiveを説明した．また，もう一つのRTOSなしでも利用できる動的アーキテクチャであるイベント駆動型については「外部に依存しすぎる」という問題があることを述べた．「外部に依存しすぎる」ということは，外部との接点が絶たれたときに，重大な問題が発生する可能性があることを意味している．

今回は，時間駆動型だけでは対応できないアプリケーションが現実には存在すること，そしてその場合にはイベント駆動型を利用するしかないが，外部に依存しすぎるという問題点をどのように解決するかについて回避策の一つを紹介する．イベント駆動型を用いる場合でも，フェイル・ソフトの考え方を導入することにより，問題が発生しても「かろうじて動き続ける」ようなシステムを構築することが可能になる．これらについても取り上げる．

1 時間駆動の問題点

● 時間駆動の構造

周期が連続的に変動する場合，Cyclic Executiveでは対応が難しいことを前回述べた．もっとも，離散的なモード変更であればそのままで対応できる．問題はタスク・セットは同一のままで周期だけを変えるような場合である．この場合はマイナ・サイクルを作り出すのに利用しているインターバル・タイマの割り込み間隔を動的に変更することなどで実現できるかもしれない．

しかし，Cyclic Executiveは静的なスケジュールを前提としているため，すでに決定済みのマイナ・サイクルを変更してしまうとデッドラインを守れる保証がなくなってしまう．マイナ・サイクルにアイドル・タイムがあれば，それを予備の保険として使い，その範囲内で対応できることもある(保険がなかった場合には対応できない)．

しかし根本的な問題として，Cyclic Executiveは外界の情報を自分から取りに行く構造なので，いつ起動周期を変更するべきかを知ることが難しい．何か適当な外界に対する物理モデルがあれば可能だが，つねにそのようなモデルが存在するとは限らない．適当なモデルがない場合には，最短周期でポーリングを行うことになるので，CPU時間をむだに消費してしまう．また，その最短周期自身が動的に変更になる場合もある(図1)．

ここでエンジンの制御を例に考えてみよう．吸気と点火のタイミングは，クランク・シャフト角に対してプロットすると図2のように周期動作になるが，時間に対してプロットすると図3のようにかなり変化する．変化の元は，ドライバのアクセルの踏み加減である．図1に示したように，時間駆動アーキテクチャでは外界の情報は状態データになるので，ドライバがいつどのような速度で踏み込んだかどうかはソフトウェアにはわからない．わかるのは，そのとき踏み込まれている深さのような情報のみである．この情報から次に起動する時刻を決めるためには，エンジンの動作に関する物理モデルが必要になる．しかも，そのときのエンジン回転数のような直接的な情報ではないので計算量や変数量が多くなり，それを少ないリソースの中で実装することは，あまり現実的ではない．

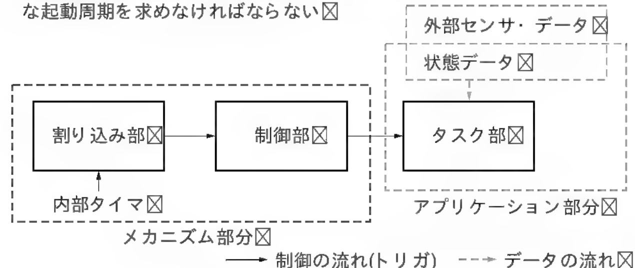


図1　時間駆動の構造

● イベント駆動の問題点

イベント駆動型の場合には，図4に示すように状態データのみでなく，タイム・スタンプ付き状態データやイベント・データを使うことができる．静的なモジュール構造は，図1も図4も割り込み部と制御部からなるメカニズム部分とアプリケーションを実装するタスク部分から構成されている．静的なアーキテクチャはほとんど同一だが，動的な構造が異なる．いうまでもなく，組み込みシステムの場合は動的な構造が重要である．

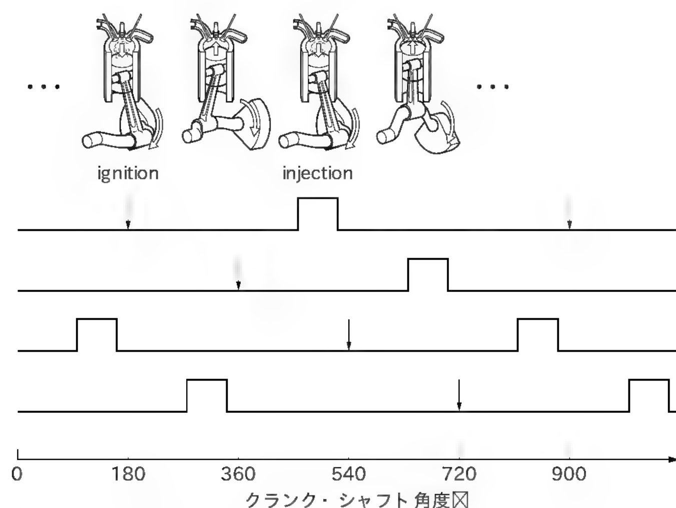


図2　エンジンのタイミング-1

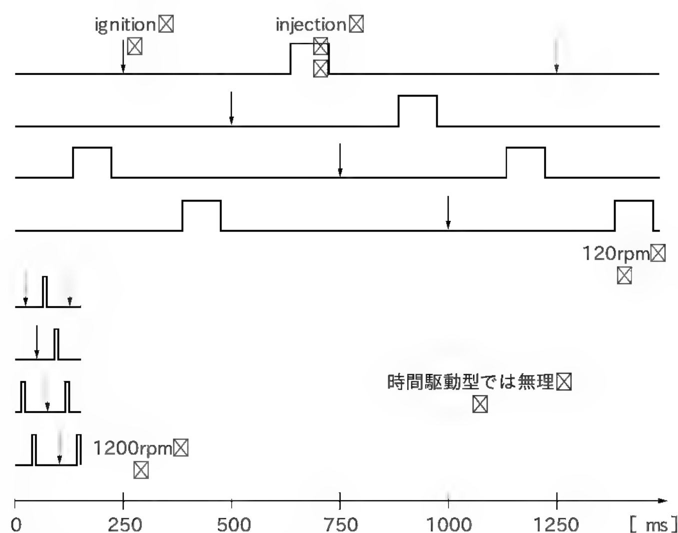


図3　エンジンのタイミング-2

つまり，UMLでいうとクラス図では表現できない部分が重要になり，これらはシーケンス図などによって表現される部分である．静的な構造から動的な制御構造を識別するためには，データに関して，「状態データか？イベント・データか？」のような「データの質」について注意する必要がある．また，データには寿命があることも認識する必要がある．

イベント駆動型の場合には，たとえばエンジンが一回転したことを検出するセンサを利用すれば制御タスクを起動できるので，自分で起動タイミングを計算する必要はなくなる．外界の事象を検出して制御タスクを起動できることがメリットである．余計なループを構成する必要がないのでCPU効率が良くなる．

ただし，今度はそこが弱点になってしまい，外界に依存しすぎる点が問題となる．どのような問題点かというと，イベントの発信元のセンサなどが故障したとき，

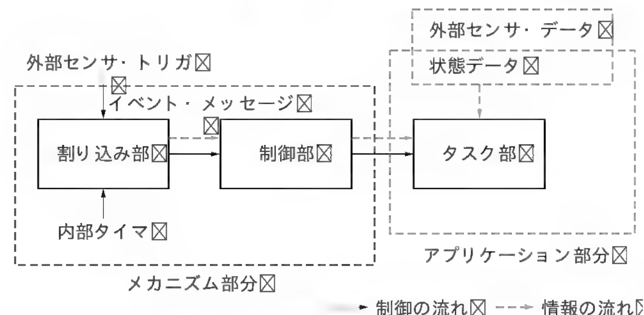


図4　イベント駆動構造

1) まったく動かなくなる
2) 逆にイベントが頻発してオーバロードになる

の二点が発生する．時間駆動型の場合は，処理ルーチンの起動は内部機構のみに依存しているので，上記のような問題は起こらない．時間駆動型ではタスクの起動は完全にシステム内部でコントロールされている．

高速走行中にエンジンの回転を検出するセンサが故障しただけで突然エンジンが止まってしまっては，事故が起きかねない．せめて安全なところまで走れるか，できればトロトロ走行でもよいので修理工場まで走りたい．どうせ作るならそんな組み込みシステムを作りたいとエンジニアなら思うだろう．

2 信頼性をどう考えるか

● ハードはいつか壊れる，ソフトは劣化しない

物理的な部品は徐々に劣化し，いずれは壊れてしまう．しかし，ソフトウェアは物理的なものではないので劣化することも擦り切れることもない．バグはもともと存在していたものなので，信頼性の中でも固有信頼度(Inherent Reliability)に属する．今，注目しているのは使用信頼度(Use Reliability)と呼ばれるものである．つまり，経時的に劣化することのないソフトウェアによってのみ，経時的に劣化するハードウェアの保全性(Maintainability)を確保してシステム全体の信頼性を高めることができるということである．先にイベント駆動型が問題だといったのは，このソフトウェアの役割を果たしていないという意味である．イベント・ドリブン型の動的アーキテクチャを利用する場合には，ハードの故障発生時に，システムが機能を完全に喪失するのではなく，ある程度の範囲で機能が維持されるようにするフェイル・ソフト(fail soft)を検討する必要がある．

ハードウェアで利用されるフェイル・セーフ(fail safe)の場合は，たとえば，原子炉では制御棒駆動装置の電源が落ちた場合，制御棒そのものの重さにより制御棒が炉内に落下し，安全に停止できるようになっている．自然落下のような物理法則を直接利用することで，システムがつねに安全側に倒れるように設計する．

ソフトウェアの場合には，物理法則を直接使うことはできな

いので，ドメイン知識としての物理法則を利用したモデルを利用する．これは，アクセルの踏み加減からエンジンの回転数を予測して最短周期を予測するのと基本的には似ているが，もっと単純な形で利用する．たとえば，「2000rpmで回転していたエンジンが突然止まることは慣性の法則から不可能である」という程度である．「エンジンは急には止まれない」ということを利用すれば，センサ入力が突然なくなったときに異常を検出できる．また，時間の概念を利用することで起こるべきことが起こらないことを検出するともいえる．

これは，イベント・データを利用することで可能になる．このためには概念モデリングをする際に，時間の概念を捉えることが必要である．このようにいうと難しいが，要するにタイム・アウトを使える部分と使えない部分を認識するということである．あるいは，「早すぎる」とか「遅すぎる」ということが起こるかどうかである．アプリケーション・ドメインにこれらの概念があれば，フェイル・ソフトを実現することができる．具体的には，「早すぎる」ことを検出して最短割り込み間隔まで遅延して動作する．同様に「遅すぎる」ことを検出したら最長割り込み間隔のところでイベント入力がなくても，処理タスクを起動すればよい．

概念モデリングの際には，オブジェクトに注意が向きがちであるが，それは静的モデリングに偏向した方法である．動的なモデルを作るためにはイベントに注目する必要がある．そして，分析の際には抽出したイベントのタイプ分けを行う．この際に，この連載の第11回で説明したイベント到着モデルを使うことができる．

図5の中で，不規則型か制限型であれば「早すぎる」と「遅すぎる」によってfail softを導入することが可能になる（図6）．

耐故障性（fault tolerance）を実現する手段としては，ドメイン知識によらずにセンサやCPUを多重化して対応する方法もあるが，これは非常にコストがかかる．また，応答がずれた場合の同期の取り方や情報をコピーして持つ場合の持ち方の問題など技術的な問題もある．以上のことより，傾向としてはドメイン知識を利用した物理モデルを使ってfail softを利用する方向にある．

3 イベント・ドリブン型の改良

イベント・ドリブン型のパフォーマンスの良さを残しながら，センサの故障に対する脆弱性を改善する方法としてTask-Splitting Model(1)というものがある．この方法では，イベント入力があってイベント処理タスクを起動すると，イベント処理タスクが最短割り込み間隔の間だけ自分自身をロックする．この間にイベント入力があっても直ちに処理されることはなく保留されて，最短割り込み間隔の間待たされる．最短割り込み間隔を経過後ロックが解除されて処理タスクが起動される．こうすることで，センサが故障して割り込みを頻発した場合でも，システムがオーバロード状態になることを防ぐことができる．この連載の第11回で紹介したデファー・サーバやスポラディック・サーバと同等のことをRTOSなしで実現したものだ．

一方，イベントが入らなくなった場合は，最長割り込み間隔を経過してもイベントが入らないことを検出してイベント処理タスクを起動することで対応する．このようにすることで，センサが壊れてもエンジンが突然止まることはなく，壊れる直前の起動周期に対応する最長割り込み間隔で動作するようになる．

具体的には，図7に示したように元のイベント処理タスクからサーバータスクSTと二つのタイミング・タスクTM1，TM2，それからタイミング予測関数f（）を作る（図8）．サーバ・タス

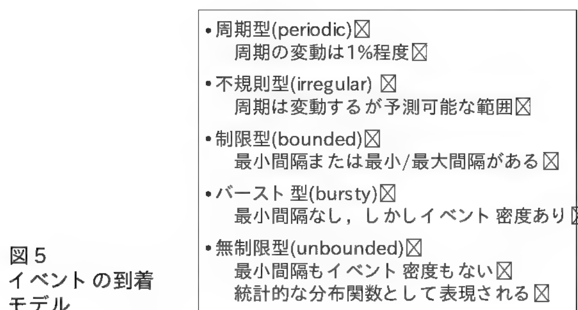


図5 イベントの到着モデル

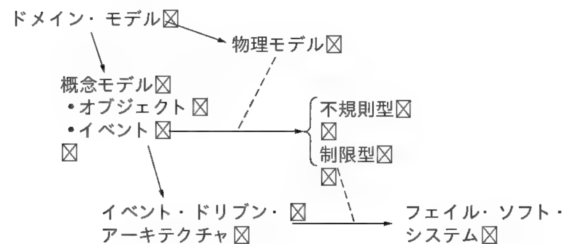


図6 fail softの実現

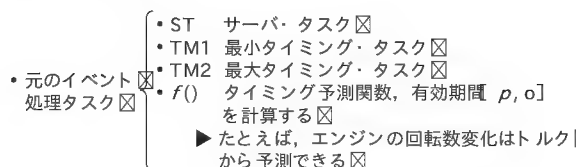


図7 Task-Splitting Model

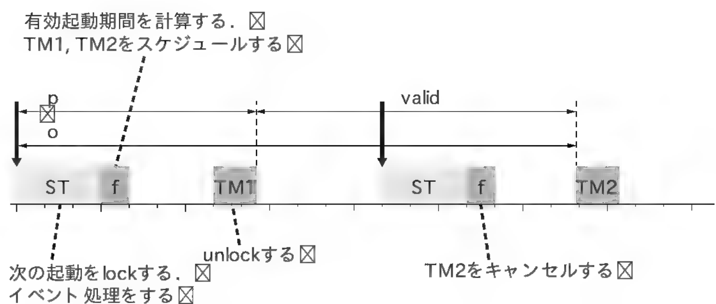


図8 各タスクの動作

クは，元のイベント処理タスクが行っていた処理のほかに，自分自身のロックとタイミング予測関数の評価とその結果に基づくタイミング・タスクの起動を設定する．*f*( )には，物理モデルに基づくその状況でのタイミングの上限値と下限値を計算するロジックを実装する(図9，図10)．

おわりに

組み込みシステムの信頼性を高めるためにフェイル・ソフトを導入することは，ソフトウェア・エンジニアだけの思い付きでは難しい．ドメイン・エンジニアやハードウェア・エンジニアなどの共同作業が必要である．組織の枠を超えた風通しの良さが必要ではないだろうか．

参考文献

(1) *S. Poledna, Reliability of Event-Triggered Task Activation for Hard Real-Time Systems.* In Proceedings of the 14th International Symposium on Real-Time Systems. IEEE Computer Society Press. 1993.

ふじくら・としゆき　(株)豆蔵

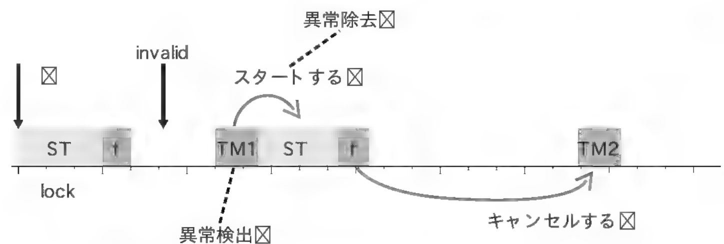


図9　早すぎる場合

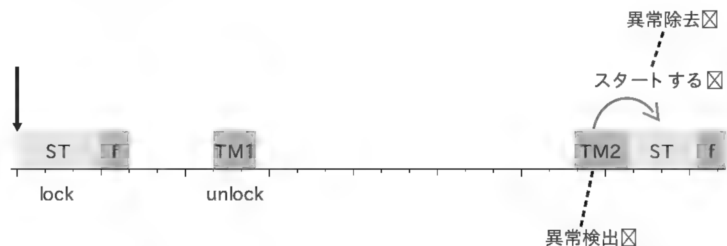


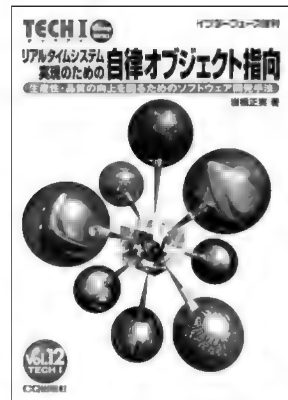
図10　遅すぎる，あるいはない場合

TECH I Vol.12　好評発売中

リアルタイムシステム実現のための 自律オブジェクト指向

生産性，品質の向上を図るためのソフトウェア開発手法

岩橋 正実 著　B5判136ページ　定価1,800円(税込)
ISBN4-7898-3323-2



さまざまな分野で，さまざまな人達が，似て非なるソフトウェアを無数に作っている．そして，ソフトウェアに対する要求は膨大・複雑化し，論理的に矛盾がないことを立証しきれない状況になってきている．また，どうすれば欠陥がないと立証できるかが重要になってきている．なかでも，リアルタイム制御システムへのオブジェクト指向の適用は，課題点とされているか，非常に不鮮明なものになっているようである．制御システムの分野では，実装資源の制約と実用的な解説が少ないため，未だ多くの問題を抱えている．このような事態を改善するために，本書では「自律オブジェクト指向」という新しい考え方を提唱する．

第1章　現状の問題点とオブジェクト指向を導入する利点
1.1　なぜオブジェクト指向をリアルタイム制御に利用するのか
1.2　リアルタイム制御システムとオブジェクト指向
1.3　自律オブジェクト指向と品質向上

第2章　自律オブジェクト指向の考え方とオブジェクトの抽出・整理
2.1　自律オブジェクト指向
2.2　状態抽出・整理テクニック

第3章　クラス/デザインパターン/フレームワークの抽出と統合化
3.1　異機種間で互換性を保つために
3.2　オブジェクトの抽出
3.3　オブジェクトの分類
3.4　クラスの抽出
3.5　フレームワークの抽出
3.6　デザインパターンの抽出
3.7　統合化モデル
3.8　まとめ

第4章　リアルタイム制御システムの分析・設計・実装・検証
4.1　分析から設計へ
4.2　実装から検証へ

第5章　オブジェクト指向で実現するリアルタイム設計
5.1　はじめに
5.2　リアルタイム制御システムの問題
5.3　リアルタイムの問題の改善
5.4　オブジェクト間の接続
5.5　オブジェクト間インターフェース構造の規定
5.6　オブジェクトの実装
5.7　イベントドリブン構造の解決策
5.8　リアルタイムOSとオブジェクト指向技術の融合
5.9　まとめ

第6章　クラスオブジェクト/サービス/割り込みの実装
6.1　非オブジェクト指向言語での実装
6.2　クラスオブジェクトの実装
6.3　クラスオブジェクトの実装の規定
6.4　サービスの実装
6.5　割り込み(タスク)の実装
6.6　まとめ

第7章　自律オブジェクト指向のデザインパターン化
7.1　ドメイン分析
7.2　デザインパターン
7.3　パターンと見積もり技術
7.4　おわりに

CQ出版社　〒170-8461 東京都豊島区巣鴨1-14-2　販売部 TEL.03-5395-2141　振替 00100-7-10665

MMUなしでも動作するLinux

はじめて使うμClinux

荘司 靖

第2回 FPGAとソフト・コアCPUでμClinuxを動かす

1 はじめに

大規模化が進みCPUさえも飲み込みはじめたFPGAは，組み込み技術者にはとても魅力的なデバイスです．今回は，そんなFPGAをベースにしたコンピュータ・ボード「SUZAKU」の紹介と，μClinux界におけるデファクト・スタンダードのディストリビューション「uClinux-dist」をベースにしたアプケーション開発を紹介します．

後半では，SUZAKUのブート・シーケンスを紹介します．電源が投入されてから，ユーザ・ランドのアプリケーションを実行するまでの過程を説明します．

また，組み込み機器に多く使われるようになってきたLinuxは，リアルタイム機能の欠如が問題視されています．コラムでは，SUZAKUならではのリアルタイム処理の解決方法を紹介します．

2 SUZAKUの概要

SUZAKU（写真1，図1）はアットマークテクノが開発した，小型コンピュータ・ボードです．CPUにはXilinxが開発したソフト・コア・プロセッサMicroBlazeを採用しています．CPUボードとしてのSUZAKUには表1のような特徴があります．

SUZAKUのCPUとしての物理的な基本構成は，SUZAKUのブロック図（図2）の破線枠内の三つです．

- FPGA

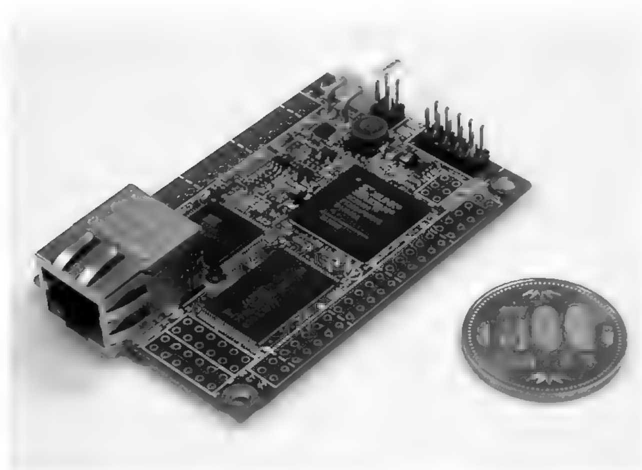


写真1 SUZAKUの外観

表1 SUZAKUの仕様

CPU	MicroBlaze（ソフト・コア）
動作周波数	51.6096MHz
メモリ	SDRAM 16Mバイト フラッシュ・メモリ 4Mバイト
Ethernet	10Base-T/100Base-TX
拡張I/Oピン	86
シリアル・ポート	1

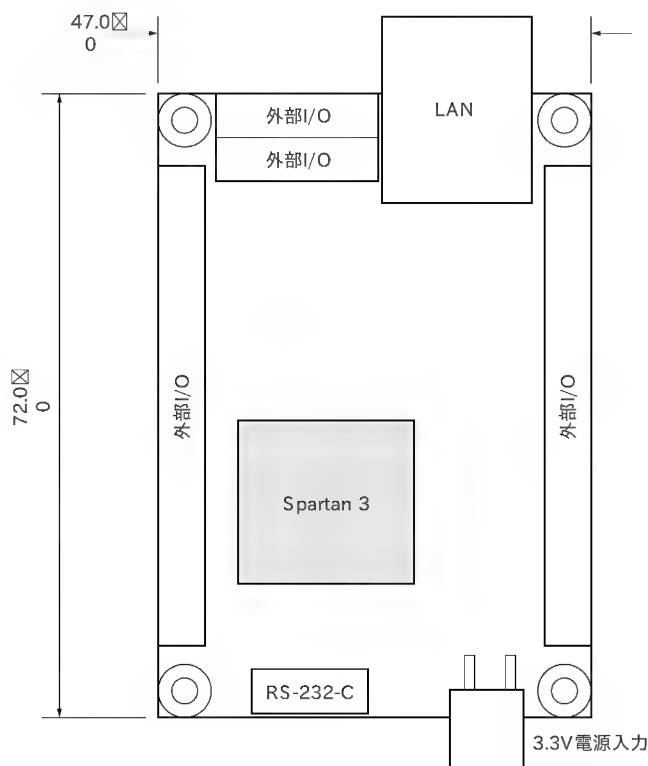


図1 SUZAKUの外形寸法

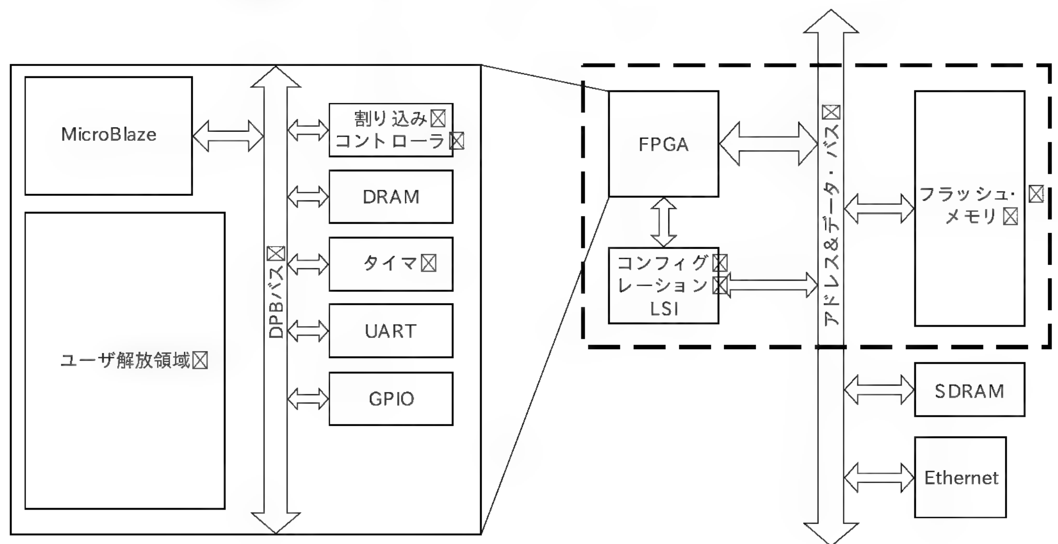


図2　SUZAKUのブロック図

- FPGA 用コンフィグレーション LSI
- FPGA のコンフィグレーション・データを格納するためのフラッシュ・メモリ

CPUボードとして必要な，CPU，割り込みコントローラ，タイマなどはすべてFPGA内部に構成されています．また，μClinuxのように多機能で比較的大規模なOSを動作させることを考慮し，16MバイトのSDRAMを搭載しています．さらに，Ethernetコントローラ(MAC/PHY)を搭載しています．Ethernetコントローラの MAC部分はFPGA内部に構成することも可能でしたが，MACが比較的複雑でFPGAのリソースを多く消費することを考慮し，外部デバイスにしています．

デフォルトでは，CPUボードとしてコンフィグレーションしています．この状態で約半分ほどのリソースがユーザ用として空いています(図3)．

```
Device utilization summary:

   Number of External IOBs            73 out of 173    42%
      Number of LOCed External IOBs   73 out of 73    100%

   Number of MULT18X18s                3 out of 16     18%
   Number of RAMB16s                   9 out of 16     56%
   Number of Slices                 1775 out of 3584   49%
      Number of SLICEMs              281 out of 1792   15%

   Number of BUFGMUXs                  1 out of 8      12%
   Number of DCMs                      1 out of 4      25%
```

図3　リソースの使用状態

3 μClinuxを動かす

μClinuxはMMUを持っていないCPUをターゲットとしたLinuxカーネルです．現在，MicroBlazeはMMUを持っていないため，SUZAKUではμClinuxを採用しています．

LinuxカーネルからARして派生したμClinuxもやはりカーネルであり，(広義の)OSではありません．OSとしての機能を果たすには，カーネル以外にシェルや`ls`のような基本的なアプリケーションが必要です．また，組み込み機器としては，機器の目的を達成するためのアプリケーションが必要になります．

ここでは，多くのμClinuxをベースにした組み込み機器で採用されているuClinux-distを使って，アプリケーションの開発方法を説明します．なるべくSUZAKUやMicroBlazeに依存しない記述となるよう配慮したつもりなので，ほかのアーキテクチャでμClinuxの開発を行うときにも参考にしてもらえればと思います．

● uClinux-dist —— The Distribution for μClinux

uClinux-distとはμClinux用のLinuxディストリビューションで，μClinuxにおいてデファクト・スタンダードな存在といっても過言ではないでしょう．

すでにデスクトップLinux向けのディストリビューションは多数ありますが，多くのものはμClinuxを使用した組み込み機器には適していません．一つの理由は，組み込み機器で使用するアプリケーションでは細かな機能のカスタマイズが必須であるという点です．汎用性を重視するPCでは多くの機能が求められますが，用途が限定された組み込みシステムでは逆に汎用性がむだになります．

また，Linux向けアプリケーションのうちの一部がμClinuxでは動作しないという点も問題です．μClinuxはMMUを持たないCPUをターゲットとしているため，Linuxの一部の機能をサポートしていません．これに該当するアプリケーションは，μClinux向けに変更を加えたり，別のアプリケーションに置き換える必要があります．

こうした背景があるため，uClinux.orgでは，カスタマイズ

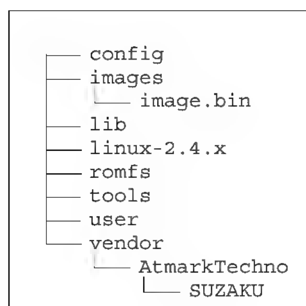


図4
ディレクトリ構成

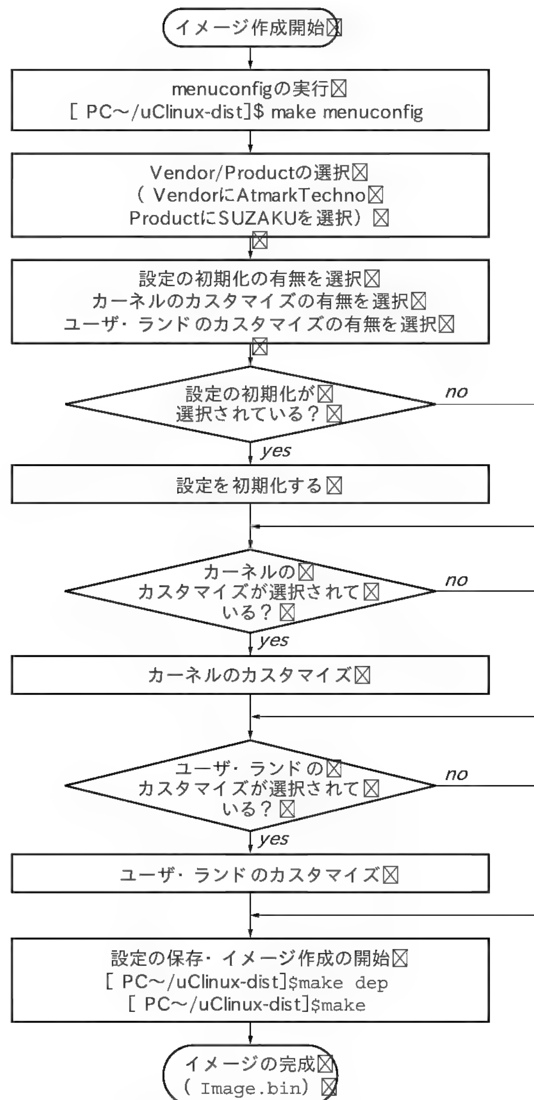


図5　コンフィグからビルドまでの流れ

性に優れたビルド・システムとμClinux用に移植されたアプリケーションを統合し，uClinux-distとして配布しています．

● uClinux-distの構成

uClinux-distについて詳しく知るには，uClinux-distがどのように構成されているかを理解する必要があります．

uClinux-distの重要なディレクトリ構成を図4に示します．

▶ `config`

ビルド・システムが使うディレクトリです．後述しますが，新しくアプリケーションを追加するときは，このディレクトリにある`config.in`を変更する必要があります．

▶ `images`

ターゲット・ボードに転送可能なバイナリ・ファイル(`image.bin`)などが生成されるディレクトリです．配布時には存在しないディレクトリで，ビルド・システムがビルドの途中で作成します．

▶ `lib`

各種ライブラリが入っています．`uClibc`や`glibc`のディレクトリは`uClinux-dist`のすぐ下にありますが，`lib`ディレクトリからシンボリック・リンクが張られています．

▶ `linux-2.4.x`

2.4系のカーネル用のディレクトリです．

▶ `romfs`

`images`と同じく，ビルド・システムがビルド中に作成するディレクトリです．ターゲット・ボードのディレクトリ構成と同じ構成が内部に作られます．

▶ `tools`

ホストPCで使用するツールが含まれています．

▶ `user`

各アプリケーションごとのディレクトリが用意されています．たとえば，ftpクライアントは，`uClinux-dist/user/ftp`となっています．自作のアプリケーションをuClinux-distのビルド・システムに組み込む場合は，この下にディレクトリを用意します．

各ボードの情報は，`vendor`ディレクトリの下に収められています．SUZAKUの場合は`uClinux-dist/vendor/AtmarkTechno/SUZAKU`となります．この`SUZAKU`のようなディレクトリをプロダクト・ディレクトリと呼びます．自社で開発したボードや，SUZAKUに大幅な変更を加えるときなどは，既存のプロダクト・ディレクトリを変更するより，新しくプロダクト・ディクレクトリを作成するほうが便利です．

`uClinux-dist`の中にはほかにもディレクトリが存在します．開発を行う前に，ぜひ覗いてみてください．

● uClinux-distのコンフィグとビルド

uClinux-distのビルド・システムはLinuxのビルド・システムを採用しているため，Linuxカーネルのビルド経験があれば直感的に使えると思います．

Linuxカーネルと同じく，以下のコンフィグ方法が使えます．

- テキスト・ベース(`config`)
- メニュー・ベース(`menuconfig`)
- GUIベース(`xconfig`)

コンフィグは大きく二つに分れています．一つがカーネルのコンフィグ，もう一つがユーザ・ランドのコンフィグです．

uClinux-distのコンフィグからビルドまでの流れを図5に示します．

● uClinux-distのユーザ・ランド・アプリケーション

ユーザ・ランド・アプリケーションのコンフィグ画面は，図6のようにいくつかのカテゴリに分類されています．これらのカテゴリを簡単に紹介します．

▶ Core Applications

システムとして動作するために必要な基本的なアプリケーションが入っています．システムの初期化を行う`init`やユーザ認証の`login`などがこのセクションで選択できます．

▶ Library Configuration

アプリケーションが必要とするライブラリの選択ができます．

▶ Flash Tools

フラッシュ・メモリに関係のあるアプリケーションが選択できます．SUZAKUでは，`netflash`と呼ばれるネットワーク・アップデート用アプリケーションがここで選択されています．

▶ Filesystem Applications

ファイル・システムに関係のあるアプリケーションが選択できます．SUZAKUでは`flatfsd`を選択しています．そのほか，`mount`，`fdisk`，`ext2`ファイル・システム，`reiser`ファイル・システム，`samba`などが含まれます．

▶ Network Applications

ネットワークに関係のあるアプリケーションが選択できます．SUZAKUで使用している`dhcpcd-new`，`ftpd`，`ifconfig`，`inetd`，`thttpd`のほかにも`ppp`やワイヤレス・ネットワークのユーティリティなども含まれます．

▶ Miscellaneous Applications

上記のカテゴリに属さないアプリケーションが収められています．UNIXの一般的なコマンド(`cp`，`ls`，`rm`など)やエディタ，オーディオ関連，スクリプト言語インタプリタなどが含まれます．

▶ BusyBox

BusyBoxのカスタマイズを行います．BusyBoxは複数のコマンド機能をもった単一コマンドで，多くの組み込みLinuxでの実績を持っています．BusyBoxはとても多くのカスタマイズができるため，別セクションになっています．

▶ Tinylogin

Tinyloginも複数コマンドの機能をもつアプケーションです．`login`や`passwd`，`getty`など認証に関係のある機能を提供します．多くのカスタマイズが可能なため別セクションになっています．

▶ Microwindows

Microwindowsは組み込み機器をターゲットにしたグラフィカル・ウィンドウ環境です．LCDなどを持つ機器を開発する場合に使えます．

▶ Games

ゲームです．説明はいらないですよね．



図6　ユーザ・ランド・アプリケーションのコンフィグ画面

▶ Miscellaneous Configuration

いろいろな設定がまとめられています．SUZAKUのrootユーザのパスワードもここで変更できます．

▶ Debug Builds

デバッグ用のオプションがまとめられています．開発中にアプリケーションのデバッグを行うときに選択します．

すべてのアプリケーションがすべてのアーキテクチャで動作することは保証されていませんが，多くの場合ほんのわずかな変更(ソース・コード・レベルであったり，`Makefile`の変更だったり)で動作させることが可能です．

● Out of Tree Compile

μClinuxで動作するオープン・ソースのアプリケーションは多数ありますが，組み込み機器の開発では，多くの場合，機器の目的に応じたアプケーションを開発する必要があります．

まず最初に，uClinux-distに変更を加えることなく，しかもuClinux-distのビルド・システムを使って手軽に開発を行える方法を紹介します．筆者らはこの方法を「Out of Tree Compile(uClinux-distのディレクトリ構造を木に見たて，そのディレクトリ外でコンパイルするため)」と呼んでいます．

ここでは説明を簡単にするために，SUZAKU用に“Hello World”を作ってみます．

まず，uClinux-distがSUZAKU用にコンフィグかつビルドされていることを確認してください．Out of Tree CompileではuClinux-distに入っているビルド・システムやライブラリ群を使うために，すでに一度ビルドされているuClinux-distが必要になります．

次に，開発するアプリケーション用のディレクトリをuClinux-distのディレクトリ構造の外に作ってください．この中には`Makefile`と必要なCのソース・コードやヘッダ・ファイルが入ります．

`hello.c`は，リスト1のようにどのCの教科書にでも出てくるような簡単なものです．`Makefile`はリスト2のようなものを使用します．この`Makefile`は“`Hello World`”以外のアプ

リスト 1　hello.c

```
#include <stdio.h>

int main(int argc, char * argv[])
{
        printf("Hello SUZAKU¥n");
        return 0;
}
```

リスト 2　Makefile（Out of Tree Compile）

```
# ROOTDIR=/usr/src/uClinux-dist ←────────── ①
ifndef ROOTDIR
ROOTDIR=../uClinux-dist
endif
ROMFSDIR  = $(ROOTDIR)/romfs
ROMFSINST = romfs-inst.sh
PATH      := $(PATH):$(ROOTDIR)/tools

UCLINUX_BUILD_USER = 1
include $(ROOTDIR)/.config
LIBCDIR = $(CONFIG_LIBCDIR)
include $(ROOTDIR)/config.arch


EXEC = hello ←────────── ②
OBJS = hello.o ←────────── ③


all: $(EXEC)

$(EXEC): $(OBJS)
    $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS)

clean:
    -rm -f $(EXEC) *.elf *.gdb *.o

romfs:
    $(ROMFSINST) /bin/$(EXEC)

%.o: %.c
    $(CC) -c $(CFLAGS) -o $@ $<
```

リスト 3　Makefile（In Tree Compile）

```
EXEC = hello
OBJS = hello.o


all: $(EXEC)

$(EXEC): $(OBJS)
    $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS)

clean:
    -rm -f $(EXEC) *.elf *.gdb *.o

romfs:
    $(ROMFSINST) /bin/$(EXEC)

%.o: %.c
    $(CC) -c $(CFLAGS) -o $@ $<
```

リスト 4　uClinux-dist/config/config.inの変更点

```
--- config.in.orig  2004-04-18 04:03:57.000000000 +0900
+++ config.in   2004-05-26 17:58:38.000000000 +0900
@@ -515,6 +515,7 @@
 bool 'gdbreplay'   CONFIG_USER_GDBSERVER_GDBREPLAY
 bool 'gdbserver'   CONFIG_USER_GDBSERVER_GDBSERVER
 bool 'hd'          CONFIG_USER_HD_HD
+bool 'hello'       CONFIG_USER_HELLO_HELLO      ← 追加した1行
 bool 'lcd'         CONFIG_USER_LCD_LCD
 bool 'ledcon'      CONFIG_USER_LEDCON_LEDCON
 bool 'lilo'        CONFIG_USER_LILO_LILO
```

リスト 5　uClinux-dist/user/Makefileの変更点

```
--- Makefile.orig       2004-02-20 13:22:55.000000000 +0900
+++ Makefile    2004-05-26 17:56:09.000000000 +0900
@@ -123,6 +123,7 @@
 dir_$(CONFIG_USER_GDBSERVER_GDBSERVER)      += gdbserver
 dir_$(CONFIG_USER_GETTYD_GETTYD)            += gettyd
 dir_$(CONFIG_USER_HD_HD)                    += hd
+dir_$(CONFIG_USER_HELLO_HELLO)              += hello  ← 追加した1行
 dir_$(CONFIG_USER_HOSTAP_HOSTAP)            += hostap
 dir_$(CONFIG_USER_HTTPD_HTTPD)              += httpd
 dir_$(CONFIG_USER_HWCLOCK_HWCLOCK)          += hwclock
```

リケーションを開発するときにもテンプレートとして使用することができます．環境に合わせて変更する点は以下の三つです．

1) ROOTDIRが指定されていない場合，現在のディレクトリと並列にuClinux-distディレクトリがあると仮定する．ほかの場所にuClinux-distがある場合は，この行のコメントを外して適切なディレクトリ名に変更する
2) 生成される実行ファイル名を指定する．ここでは，helloとする
3) 生成される実行ファイルが依存するオブジェクト・ファイルを指定する．ここではhello.oを指定する

Makefileとhello.cが用意できたら，helloをビルドします．ビルドにはmakeコマンドを使用します．ビルドが完了すると実行ファイルhelloがディレクトリ内に生成されています．

この実行ファイルをμClinuxのromfsディレクトリに配置するために，makeコマンドでromfsターゲットを指定します．

make romfsの後uClinux-distのディレクトリに移動して，make imageを実行することで，helloを含んだSUZAKU用のイメージ・ファイルがimagesディレクトリに生成されます．

● In Tree Compile

テスト・プログラムを作成する場合は，Out of Tree Compileでも問題ありませんが，多くの自作アプリケーションをSUZAKU用にビルドするには少々手間がかかります．

そこで，自作のアプリケーションをuClinux-distに統合する方法を紹介します．uClinux-distに統合してしまえば，uClinux-distのディレクトリ内でmakeするだけで，imagesディレクトリにターゲット・ボード用のイメージ・ファイルを生成することができます．

1) ディレクトリを用意する

uClinux-dist/user以下に，アプリケーション用のディレクトリを作成します．ここでは，helloとします．

2) ソース・コードとMakefileを用意する

Cのソース・コードは前項で使ったhello.cを使用します．Makefile（リスト3）はOut of Tree Compileで使用したものよりもシンプルになります．

3) uClinux-distに追加アプリケーションを教える



図7　メニューに追加されたhello

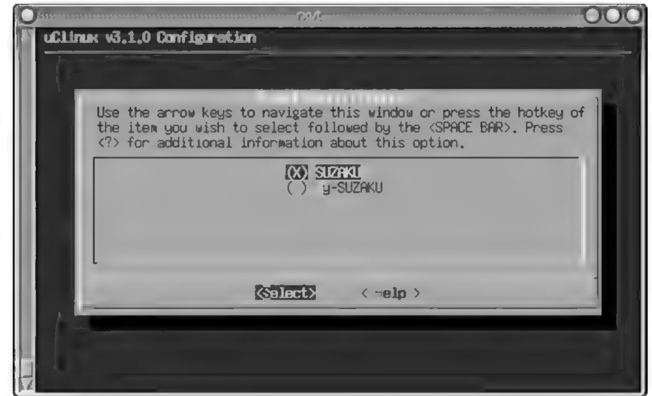
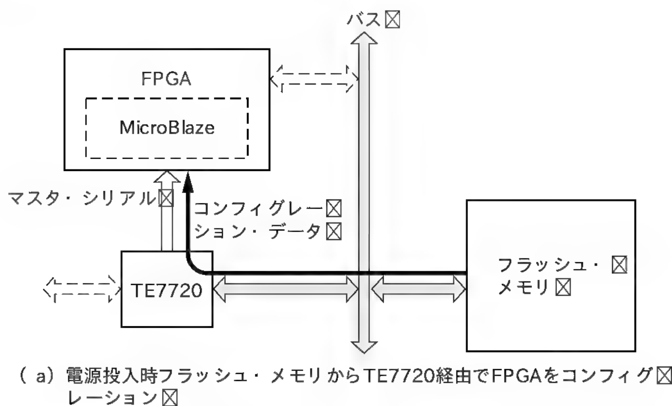
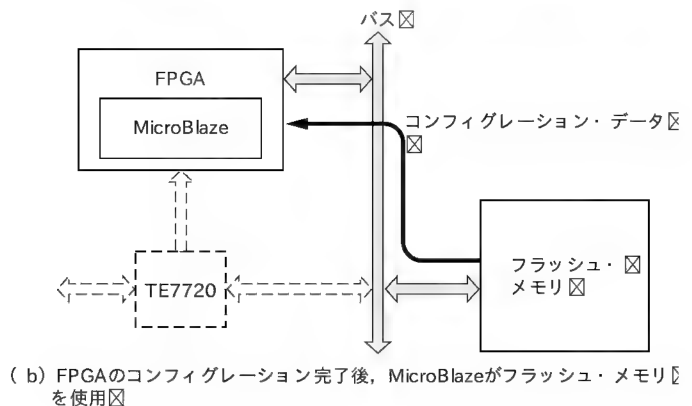


図8　新プロダクト



(a) 電源投入時フラッシュ・メモリからTE7720経由でFPGAをコンフィグレーション



(b) FPGAのコンフィグレーション完了後，MicroBlazeがフラッシュ・メモリを使用

図9　FPGAのコンフィグレーション

変更箇所は，`uClinux-dist/config/config.in`（リスト4）と`uClinux-dist/user/Makefile`（リスト5）です．この例では，追加するアプリケーションをMiscellaneous Applicationに追加します．ほかにならってアルファベット順に並べるほうが良いと思います．

4) 追加したアプリケーションを選択する

`make menuconfig`などで追加したアプリケーションがMiscellaneous Applicationセクションに表示されるか確認してください．

メニュー画面で`hello`を選択しビルドすると，生成された`image.bin`に`hello`が追加されています（図7）．ターゲット・ボードに`image.bin`を転送して，確認してください．

● プロダクト・ディレクトリ

`vendor`ディレクトリの下に各製品ごとのディレクトリがあることは前述したとおりです．SUZAKUの場合，このプロダクト・ディレクトリは`uClinux-dist/vendor/AtmarkTechno/SUZAKU`となっています．このディレクトリ配下には，製品のデフォルト設定情報や`Makefile`，起動スクリプトなどが置かれています．

アプリケーションの追加だけであれば，In Tree Compileを行うだけで可能ですが，起動スクリプトや設定情報の大幅な変更を行う場合は，新しいプロダクト・ディレクトリを作成することをお勧めします．

新しいプロダクトの作成はとても簡単です．`uClinux-dist/vendor/AtmarkTechno/`の下にある`SUZAKU`ディレクトリを新しい名前（たとえば`my-SUZAKU`など）でコピーするだけで，μClinuxのビルド・システムがメニューの中に新しいプロダクトとして表示してくれます（図8）．

この`my-SUZAKU`を開発のベースにすることで，オリジナルのSUZAKUとの`diff`も簡単に取れますし，新しいuClinux-distがリリースされたときもディレクトリをコピーするだけで簡単に追従することができます．

4 SUZAKUのブート・シーケンス（ハードウェア・ステージ）

SUZAKUのブート・シーケンスは，ハードウェア・ステージと四つのソフトウェア・ステージに分かれています．

図9にSUZAKUのハードウェアのブート・シーケンスを示します．FPGAがプログラムされた機能どおりに動作するにはコンフィグレーションが必要です．コンフィグレーションは，SUZAKUの電源回路，FPGA，コンフィグレーションLSI（TE7720）が協調して行われます．このコンフィグレーション

によって，FPGA 内部に CPU やそのほかのペリフェラルが形成され，CPU MicroBlaze がアドレス0番地から実行を始めます．

5 SUZAKU のブート・シーケンス（ソフトウェア・ステージ）

● 第1ステージ（BBoot）

MicroBlaze はリセット時にプログラム・カウンタが0に初期化されます．SUZAKU では0番地に FPGA の内部メモリがマップされており，BBoot という第1ステージ・ブート・ローダが入っています．

BBoot は第2ステージ・ブート・ローダを起動する役目と，モトローラS形式でダウンロードした第2ステージ・ブート・ローダをフラッシュ・メモリに書き込む機能をもっています．FPGA の内部メモリは非常に高価なリソースなので，できるだけ小さなプログラムにし，汎用的な高機能ブート・ローダをフラッシュ・メモリに入れることにしました．

しかし，第2ステージのブート・ローダが壊れてしまった場合に備え，モトローラS形式のダウンロード機能をもたせています．

BBoot は SUZAKU の起動モード選択ジャンパを調べ，モトローラS形式のダウンロードを行うのか，または第2ステージ・ブート・ローダを起動するのかを判断します．

● 第2ステージ（Hermit）

第2ステージ・ブート・ローダには Hermit が入っています．Hermit は ARM CPU ボード Armadillo にも採用している高機能ブート・ローダ兼ダウンローダです．

Hermit の一番大きな仕事はカーネルをブートすることです．Hermit が起動モード選択ジャンパを調べ，カーネルをブートするのかブート・ローダ・モードで起動するのかを判断します．

起動モード選択ジャンパがオープンだった場合は，カーネルイメージをフラッシュ・メモリから SDRAM にコピーしてカーネルに制御を渡します．

Hermit をブート・ローダ・モードで起動した場合（BBoot の起動モード選択画面から選択可能），Hermit は独自のバイナリ転送方式で μClinux のイメージをフラッシュ・メモリに書き込むことができます．

なお，Hermit 以外のブート・ローダを使用することも可能です．現在は U-Boot の移植も行われており，将来的に公開する予定です．

● 第3ステージ（カーネル）

カーネルはブート・ローダから制御を渡された後，システムの初期化を行います．システムの初期化の多くは一般的な Linux と同じです．スケジューリングに必要なタイマの初期化や割り込みベクタの初期化，メモリ・マネージメント・サブシステムによる有効な RAM 領域の初期化などが行われます．そして，カーネルは最後にルート・ファイル・システムをマウントしてユーザ・ランドに制御を渡します．

μClinux 特有の機能として，カーネル・イメージの最後尾にルート・ファイル・システムを結合することができます（図10）．これは m68knommu アーキテクチャが実装した機能で，現在では m68knommu のほかに armnommu と microblaze アーキテクチャで実装されています．

ROM の使用量を抑えるために，ルート・ファイル・システムはカーネルの BSS セクションを上書きするように結合されています．カーネルは初期化時にルート・ファイル・システムのイメージを発見すると，BSS セクションの後に移動させます．その後，BSS セクションを0で初期化するようになっています．

● 第4ステージ（ユーザ・ランド）

カーネルはデフォルトで最初に /sbin/init を実行します．

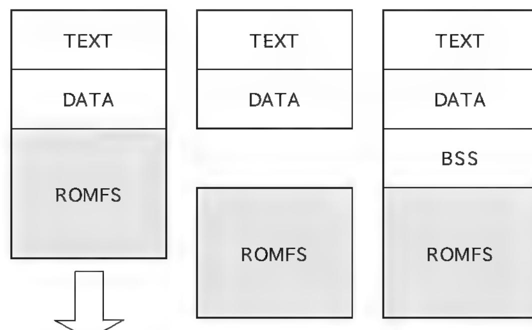


図10　結合されたルート・ファイル・システム

COLUMN

自作の CPU も夢じゃない

プログラマにとって CPU というものは，「つねにあるもの」ではないでしょうか．少なくとも筆者にとってはそうでした．ソフトウェアというものは CPU がフェッチして，実行するもので，CPU がなければソフトウェアの存在自体が無意味といっても良いのではないでしょうか——FPGA の可能性は，そんな CPU（ソフトウェアにとっては神様みたいに手の届かないもの）ですら，「プログラム」という行為をもって変更あるいは，完全なる創造すら可能にさせてくれます．

もちろん，CPU のように複雑なロジックは簡単には作れないことは重々承知しています．しかし，プログラマにとって「プログラムできる」ということは，やればできるという錯覚を与えてくれます．ここで重要なのは，「錯覚を与えてくれる」ということです．無意識のうちに人間は，「自分にはできない」とか「自分の世界とは違う」という壁を作っていることがあります．

「プログラム可能」という言葉には，それを取り去ってくれる魔力があるように思います．

/sbin/initは/etc/inittabに従ってシリアル・コンソールからのログイン用にgettyを起動したり，システムが機能するための設定やデーモンの起動を行います．使用している/sbin/initの実装はデスクトップやサーバ用のディストリビューションとは異なりますが，シェル・スクリプトを順次実行するという基本的な動作は同じです．すでにLinuxの解説書は多く出回っているので，initのふるまいについては割愛します．

おわりに

かけ足でSUZAKUの紹介とuClinux-distをベースにした開発について紹介してきました．

SUZAKUの最大のおもしろさは，プログラムによってソフトウェアからハードウェアまで自由に変更できる点だと，筆者は思っています．もちろんソフトウェアとハードウェアの両方を理解して設計することはたいへんなことですが，そのたいへんさを上回る楽しさや嬉しさ，感動が，その先にあると信じています．

また，SUZAKUでは，多くのオープン・ソースのソフトウェアを採用しています．これらのソフトウェアを開発，メンテナンスしている世界中のプログラマに，この場を借りて感謝したいと思います．

μClinuxは，D. Jeff Dionne氏やGreg Ungere氏，David McCulloughu氏，さらにμClinux Development Listに参加しているすべての人々の成果によって支えられています．uClibc，BusyBoxは，Eric Andersen氏とBusyBox Mailing Listに参加しているすべての人々の成果によって支えられています．MicroBlazeプロセッサ・アーキテクチャへのμClinuxオリジナルポートは，John Williams氏によるものです．

COLUMN

リアルタイム処理のサポート

順調に組み込み機器の世界でも成果を出しているLinuxですが，やはりほかの組み込み用OS，特にリアルタイムOSの占めるシェアは大きいようです．

組み込みの世界では，やはりリアルタイム処理が必須項目のようですが，それでも多機能なLinuxを組み込み機器で使いたいという要望が多くあります．このため，Linuxの応答性を改善するために，Linux自身にさまざまな改良が加えられています．また，リアルタイムOSとLinuxを組み合せ，ハイブリットOSとして組み込むケースもあるようです．

SUZAKUでは，以下の二つの方法でリアルタイム処理のサポートを行います．

● もう一つのCPU

リアルタイム性が必要な処理専用に，新たにCPUを追加します．意外に思うかもしれませんが，SUZAKUのCPUはもともとFPGAの内部に構成されています．新しくCPUを追加することもFPGAの容量が許すかぎり，いくつでも可能です．

● ハードウェア・ロジック

ソフトウェアの改善のみでは必要なリアルタイム性を得られない場合もあります．多くの場合，もっと速いCPUを採用したり，専用の回路を外部に作成して対応すると思います．必要な回路にもよりますが，SUZAKUではFPGA内部に構成することで，外部回路を作成する手間を省くことができます．

参考文献，URL

(1) Drabik, John.“*World's First Embedded Linux Distribution Keeps on Growing.*”，http://www.arcturusnetworks.com/Docs/AboutuClinux.pdf
(2) MicroBlaze uClinux Project Home Page，http://www.itee.uq.edu.au/~jwilliams/mblaze-uclinux/
(3) SUZAKU Official Site，http://suzaku.atmark-techno.com/
(4) SUZAKU Hardware manual，http://suzaku.atmark-techno.com/download/suzaku/manual/suzaku_hardware_manual_ver1_0.pdf
(5) SUZAKU Software manual，http://suzaku.atmark-techno.com/download/suzaku/manual/suzaku_software_manual_ver1_0.pdf
(6) uClinux - Embedded Linux/Microcontroller Project，http://www.uclinux.org/
(7) Xilinx, Inc.，http://www.xilinx.com/
(8) WikiPedia，http://www.wikipedia.org/
(9) 宮崎 仁；CPLD/FPGAの基礎，Interface，2001年11月号，pp.63-74，CQ出版(株)．
(10) 荒井 航平，井倉 将実；FPGA/CPLDの基礎と最新動向，FPGA活用チュートリアル，pp.4-11，CQ出版(株)．
(11) 浅井 剛；ソフト・マクロのCPUを使おう!，FPGA活用チュートリアル，pp.33-40，CQ出版(株)．
(12) 特許庁総務部技術調査課；プログラマブル・ロジック・デバイス技術に関する特許出願技術動向調査，http://www.jpo.go.jp/shiryou/pdf/gidou-houkoku/pld.pdf

しょうじ・やすし　(株)アットマークテクノ

Embedded UNIX　好評発売中
組み込みエンジニアのための
Embedded UNIX Vol.6
A4変型判　定価1,490円(税込)
● 第1特集　ゼロから始めるLinuxシステム構築
● 第2特集　Linuxワンボード・コンピュータ開発記
その他，連載記事，解説記事，ニュース，技術情報満載!
CQ出版社　〒170-8461 東京都豊島区巣鴨1-14-2　販売部 TEL.03-5395-2141　振替 00100-7-10665



第18回
GCC2.95から追加変更のあったオプションの補足と検証（その6）

岸 哲夫

Fedora2がリリースされ，カーネル2.6が採用されました．CPUも，日を追うごとに高速化していますが，よりいっそうコードの効率化も要求されることでしょう．
今回は，引き続きGCC2.95から追加変更のあったオプションの補足と検証を行います．重要な「最適化オプション」について扱います．（筆者）

● -fgcse，-fgcse-lm，-fgcse-sm

共通部分式の除去やコピー伝播の技法を用いた最適化を行います．-fgcse-lmの場合，それに加えて意味のないループ内のストアやロードを移動し，それをループの外側に追い出すことを試行します．-fgcse-smの場合，ロードやストアを含んでいるループはそのループの前後に移動することを試みます．

このオプションは単独で使用して，その効果を明示することが困難なのでプログラム例は略します．

● -fif-conversion，-fif-conversion2

アセンブラに展開された状態で生成される，コンディション・フラグによる条件分岐を最適化します．

このオプションもまた単独で使用して，その効果を明示することが困難なのでプログラム例は略します．

● -finline-functions

このオプションを指定すると単純なstatic関数をインライン関数に置き換えます．このオプションは-O3オプションとともに指定します．

ソースと生成されたコードを，リスト1～リスト3に示します．

この最適化の結果，関数test1の呼び出しが，実際の命令に置き換えられています．何度もtest1を呼び出す場合，コードのサイズが大きくなりますが，関数呼び出しにかかるオーバヘッド時間が減少し，処理が速くなることがあります．

実行結果は次のようになります．

```
$ gcc test217.c -o test217
$ ./test217
1
2
$
```

このオプションでインライン指定をするのも良いと思いますが，キーワードinlineを指定したほうが可読性も高まり，可

リスト1 -finline-functionsオプションを使う例（test217.c）

```
//インラインファンクションの例
#include <stdio.h>
static int    a;
static int    test1();
static int    test1()
{
    return    a++;
}
int main()
{
    int       res;
    a    =    1;
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    return    0;
}
```

リスト2 -finline-functionsオプションを付けて生成されたアセンブラ・ソース（test217a.s）

```
    .file     "test217.c"
    .section  .rodata.str1.1,"aMS",@progbits,1
.LC0:
    .string   "%d¥n"
    .text
    .p2align 2,,3
.globl main
    .type     main,     @function
main:
    pushl     %ebp
    movl      %esp,     %ebp
    subl      $8,       %esp
    andl      $-16,     %esp
    subl      $8,       %esp
    pushl     $1
    pushl     $.LC0
    movl      $2,       a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    xorl      %eax,     %eax
    leave
    ret
    .size     main,     .-main
    .local    a
    .comm     a,4,4
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト 3　-fno-inline-functions オプションを付けて生成されたアセンブラ・ソース(test217.s)

```
    .file     "test217.c"
    .text
    .p2align  2,,3
    .type     test1,    @function
test1:
    movl      a,        %eax
    pushl     %ebp
    movl      %esp,     %ebp
    leal      1(%eax),  %ecx
    movl      %ecx,     a
    leave
    ret
    .size     test1,    .-test1
    .section  .rodata.str1.1,"aMS",@progbits,1
.LC0:
    .string   "%d¥n"
    .text
    .p2align  2,,3
.globl main
    .type     main,     @function
main:
    pushl     %ebp
    movl      %esp,     %ebp
    subl      $8,       %esp
    andl      $-16,     %esp
    movl      $1,       a
    call      test1
    subl      $8,       %esp
    pushl     %eax
    pushl     $.LC0
    call      printf
    call      test1
    popl      %edx
    popl      %ecx
    pushl     %eax
    pushl     $.LC0
    call      printf
    xorl      %eax,     %eax
    leave
    ret
    .size     main,     .-main
    .local    a
    .comm     a,4,4
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

搬性も高まると思います．

ソースと生成されたコードを，リスト 4，リスト 5に示します．

実行結果は次のようになります．もちろん先と同じ結果になります．

```
$ gcc test218.c -o test218
$ ./test218
1
2
$
```

● -finline-limit=n

このオプションを付加するとインライン関数を展開する数を指定できます．つまりn以上のインライン関数があった場合，通常の関数として扱うようになります．なお，デフォルト値は600になっています．

ソースと生成されたコードを，リスト 6～リスト 8に示します．

リスト 8からわかるように，5以上あるインライン関数は無視されています．

● -fkeep-inline-functions

このオプションを付加すると，対象の関数に対する呼び出しが全部統合され，その関数がstaticと宣言されている場合でも，実行時に呼び出し可能な関数を別個に出力します．このオプションは，extern inline宣言された関数には影響しません．

このオプションに関しては，意図しない動作を招くかもしれ

リスト 4　キーワード inlineを指定した例(test218.c)

```
//インライン・ファンクションの例
#include <stdio.h>
static int    a;
static inline int test1();
static inline int test1()
{
    return    a++;
}
int main()
{
    int       res;
    a    =    1;
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    return    0;
}
```

リスト 5　-O3オプションを付けて生成されたアセンブラ・ソース(test218.s)

```
    .file     "test218.c"
    .section  .rodata.str1.1,"aMS",@progbits,1
.LC0:
    .string   "%d¥n"
    .text
    .p2align  2,,3
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp,     %ebp
    subl      $8,       %esp
    andl      $-16,     %esp
    subl      $8,       %esp
    pushl     $1
    pushl     $.LC0
    movl      $2,       a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    xorl      %eax,     %eax
    leave
    ret
    .size     main,     .-main
    .local    a
    .comm     a,4,4
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト 6　-finline-limit=オプションを使う例(test219.c)

```
//インライン・ファンクションの例
#include <stdio.h>
static  int  a;
static  int  test1();
static  int  test1()
{
    return    a++;
}
int main()
{
    int       res;
    a    =    1;
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    return    0;
}
```

リスト 7　n=200で生成されたアセンブラ・ソース(test219a.s)

```
    .file     "test219.c"
    .section  .rodata.str1.1,"aMS",@progbits,1
.LC0:
    .string   "%d¥n"
    .text
    .p2align 2,,3
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp,     %ebp
    subl      $8,       %esp
    andl      $-16,     %esp
    subl      $8,       %esp
    pushl     $1
    pushl     $.LC0
    movl      $2,       a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %ecx
    popl      %eax
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %ecx
    popl      %eax
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %ecx
    popl      %eax
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %ecx
    popl      %eax
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %ecx
    popl      %eax
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    xorl      %eax,     %eax
    leave
    ret
    .size     main,     .-main
    .local    a
    .comm     a,4,4
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト 8　n=5で生成されたアセンブラ・ソース(test219.s)

```
    .file     "test219.c"
    .section  .rodata.str1.1,"aMS",@progbits,1
.LC0:
    .string   "%d¥n"
    .text
    .p2align 2,,3
.globl main
    .type     main,     @function
main:
    pushl     %ebp
    movl      %esp,     %ebp
    subl      $8,       %esp
    andl      $-16,     %esp
    movl      $1, a
    call      test1
    subl      $8,       %esp
    pushl     %eax
    pushl     $.LC0
    call      printf
    call      test1
    popl      %edx
    popl      %ecx
    pushl     %eax
    pushl     $.LC0
    call      printf
    call      test1
    popl      %edx
    popl      %ecx
    pushl     %eax
    pushl     $.LC0
    call      printf
    call      test1
    popl      %edx
    popl      %ecx
    pushl     %eax
    pushl     $.LC0
    call      printf
    call      test1
    popl      %edx
    popl      %ecx
    pushl     %eax
    pushl     $.LC0
    call      printf
    call      test1
    popl      %edx
    popl      %ecx
    pushl     %eax
    pushl     $.LC0
    call      printf
    call      test1
    popl      %edx
    popl      %ecx
    pushl     %eax
    pushl     $.LC0
    call      printf
    call      test1
    popl      %edx
    popl      %ecx
    pushl     %eax
    pushl     $.LC0
    call      printf
    call      test1
    popl      %edx
    popl      %ecx
    pushl     %eax
    pushl     $.LC0
    call      printf
    call      test1
    popl      %edx
    popl      %ecx
    pushl     %eax
    pushl     $.LC0
    call      printf
    call      test1
    popl      %edx
    popl      %ecx
    pushl     %eax
    pushl     $.LC0
    call      printf
    xorl      %eax,     %eax
    leave
    ret
    .size     main,     .-main
    .local    a
    .comm     a,4,4
    .p2align 2,,3
    .type     test1,    @function
test1:
    movl      a,        %eax
    pushl     %ebp
    movl      %esp,     %ebp
    leal      1(%eax),  %ecx
    movl      %ecx,     a
    leave
    ret
    .size     test1,    .-test1
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト 9　-fkeep-inline-functions オプションを使う例(test220.c)

```
//インライン・ファンクションの例
#include <stdio.h>
static  int  a;
static  int  test1();
static  int  test1()
{
    return    a++;
}
int main()
{
    int       res;
    a    =    1;
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    return    0;
}
```

リスト 10　-fkeep-inline-functions で生成されたアセンブラ・ソース(test220a.s)

```
    .file     "test220.c"
    .section  .rodata.str1.1,"aMS",@progbits,1
.LC0:
    .string   "%d¥n"
    .text
    .p2align 2,,3
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp,     %ebp
    subl      $8,       %esp
    andl      $-16,     %esp
    subl      $8,       %esp
    pushl     $1
    pushl     $.LC0
    movl      $2,       a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %ecx
    popl      %eax
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %ecx
    popl      %eax
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %ecx
    popl      %eax
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %ecx
    popl      %eax
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %ecx
    popl      %eax
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    xorl      %eax,     %eax
    leave
    ret
    .size     main,     .-main
    .local    a
    .comm     a,4,4
    .p2align  2,,3
    .type     test1,    @function
test1:
    movl      a,        %eax
    pushl     %ebp
    movl      %esp,     %ebp
    leal      1(%eax),  %ecx
    movl      %ecx,     a
    leave
    ret
    .size     test1,    .-test1
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト 11 -fno-keep-inline-functions で生成されたアセンブラ・ソース(test220.s)

```
    .file     "test220.c"
    .section  .rodata.str1.1,"aMS",@progbits,1
.LC0:
    .string   "%d¥n"
    .text
    .p2align 2,,3
.globl main
    .type     main,     @function
main:
    pushl     %ebp
    movl      %esp,     %ebp
    subl      $8,       %esp
    andl      $-16,     %esp
    subl      $8,       %esp
    pushl     $1
    pushl     $.LC0
    movl      $2,       a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %ecx
    popl      %eax
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %ecx
    popl      %eax
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %ecx
    popl      %eax
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %ecx
    popl      %eax
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %ecx
    popl      %eax
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    popl      %eax
    popl      %edx
    movl      a,        %ecx
    pushl     %ecx
    leal      1(%ecx),  %edx
    pushl     $.LC0
    movl      %edx,     a
    call      printf
    xorl      %eax,     %eax
    leave
    ret
    .size     main,     .-main
    .local    a
    .comm     a,4,4
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

ないので注意してください．

ソースと生成されたコードをリスト 9～リスト 11に示します．

● -fkeep-static-consts

このオプションはデフォルトで付加されます．これは static const 宣言された変数を，たとえその変数が参照されていなくても出力するようにします．-fno-keep-static-consts オプションを使うと参照されていない static const 変数は出力されません．

ソースと生成されたコードをリスト 12～リスト 16に示します．

参照されていない static const 変数「test_static」は，

リスト 12 -fkeep-static-consts オプションを使う例(test221.c)

```
//fkeep-static-consts の例
#include <stdio.h>
static const test_static;
int main()
{
    int      res;
    int      a    =    1;
    return   0;
}
```

リスト 13　-fkeep-inline-functions で生成されたマップ・リスト(test221.nm)

```
08049448 D _DYNAMIC
08049514 D _GLOBAL_OFFSET_TABLE_
0804842c R _IO_stdin_used
         w _Jv_RegisterClasses
08049438 d __CTOR_END__
08049434 d __CTOR_LIST__
08049440 d __DTOR_END__
0804943c d __DTOR_LIST__
08048430 r __FRAME_END__
08049444 d __JCR_END__
08049444 d __JCR_LIST__
08049530 A __bss_start
08049524 D __data_start
080483e8 t __do_global_ctors_aux
080482d4 t __do_global_dtors_aux
08049528 D __dso_handle
08049434 A __fini_array_end
08049434 A __fini_array_start
         w __gmon_start__
08049434 A __init_array_end
08049434 A __init_array_start
080483a4 T __libc_csu_fini
0804835c T __libc_csu_init
         U __libc_start_main@@GLIBC_2.0
08049434 A __preinit_array_end
08049434 A __preinit_array_start
08049530 A _edata
08049538 A _end
0804840c T _fini
08048428 R _fp_hw
08048254 T _init
0804828c T _start
080482b0 t call_gmon_start
08049530 b completed.1
08049524 W data_start
08048310 t frame_dummy
0804833c T main
0804952c d p.0
08049534 b test_static
```

リスト 14　-fno-keep-inline-functions で生成されたマップ・リスト(test221a.nm)

```
08049448 D _DYNAMIC
08049514 D _GLOBAL_OFFSET_TABLE_
0804842c R _IO_stdin_used
         w _Jv_RegisterClasses
08049438 d __CTOR_END__
08049434 d __CTOR_LIST__
08049440 d __DTOR_END__
0804943c d __DTOR_LIST__
08048430 r __FRAME_END__
08049444 d __JCR_END__
08049444 d __JCR_LIST__
08049530 A __bss_start
08049524 D __data_start
080483e8 t __do_global_ctors_aux
080482d4 t __do_global_dtors_aux
08049528 D __dso_handle
08049434 A __fini_array_end
08049434 A __fini_array_start
         w __gmon_start__
08049434 A __init_array_end
08049434 A __init_array_0start
080483a4 T __libc_csu_fini
0804835c T __libc_csu_init
         U __libc_start_main@@GLIBC_2.0
08049434 A __preinit_array_end
08049434 A __preinit_array_start
08049530 A _edata
08049534 A _end
0804840c T _fini
08048428 R _fp_hw
08048254 T _init
0804828c T _start
080482b0 t call_gmon_start
08049530 b completed.1
08049524 W data_start
08048310 t frame_dummy
0804833c T main
0804952c d p.0
```

リスト 15　-fkeep-inline-functions で生成されたアセンブラ・ソース(test221.s)

```
    .file    "test221.c"
    .text
.globl main
    .type    main,    @function
main:
    pushl    %ebp
    movl     %esp,    %ebp
    subl     $8,      %esp
    andl     $-16,    %esp
    movl     $0,      %eax
    subl     %eax,    %esp
    movl     $1,      -8(%ebp)
    movl     $0,      %eax
    leave
    ret
    .size    main,    .-main
    .local   test_static
    .comm    test_static,4,4
    .section .note.GNU-stack,"",@progbits
    .ident   "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト 16　-fno-keep-inline-functions で生成されたアセンブラ・ソース(test221a.s)

```
    .file    "test221.c"
    .text
.globl main
    .type    main,    @function
main:
    pushl    %ebp
    movl     %esp,    %ebp
    subl     $8,      %esp
    andl     $-16,    %esp
    movl     $0,      %eax
    subl     %eax,    %esp
    movl     $1,      -8(%ebp)
    movl     $0,      %eax
    leave
    ret
    .size    main,    .-main
    .section .note.GNU-stack,"",@progbits
    .ident   "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

-fno-keep-inline-functions オプションを付けるとマップ・リスト(リスト 14)にも生成されたアセンブラ(リスト 16)にも出力されていません.

コードのサイズを極限まで小さくしたいときに使用すると便利です.

＊　＊

次回も引き続き「最適化オプション」の補足を行います.

きし・てつお

ハッカーの常識的見聞録



今月の常識

Virtual Server 2005 RC版をテストしよう

広畑 由紀夫

☆今月はいろいろと話題の多い中，今後のサーバ仮想化技術の一つとして発表となったMicrosoft Virtual Server 2005(以下，Virtual Server)RC版について紹介します．

● Virtual Server 2005

Virtual Serverは，単独アプリケーションとしてのVirtual PC 2004(以下，Virtual PCと略)をサービスとして発展させた製品といえるでしょう．具体的には，Virtual PCと異なり，サービスとして起動時からバックグラウンド実行することで，ホストPCでの操作とサーバ・システムの動作のバランスを保ち，ホストPCでログインすることなく，動作させ続けることが可能であるという点です．

● Virtual PCとの違い

Virtual PCでは，MS-DOS，Windows95，Windows NT，Solarisなど他OSをゲストOSとし，仮想ハードディスクを記録媒体としてインストール，および起動して使用します．このときアプリケーション起動のため，ホストPCにログインしたうえでVirtual PCを起動しなければならないという制限が加わります．

Virtual Serverでは，サービス起動により，こうした制限が取り払われていますが，バックグラウンド・サービスの特性上，アプリケーションとは異なる制御コンソールを用いることになります．

● Virtual Server RC版の導入テスト

現在，マイクロソフト社に対してベータ・プログラムへの参加申し込みをすることで，RC版(期間制限付き)を導入，テストすることができるようになっています．また，Virtual Serverのサイトに，おもに英語ではありますが，すでに多数の技術情報が掲載されているので，一度目を通しておくことを勧めます．

もちろん，Virtual Server単体でも導入できるのですが，初めて仮想サーバに触れるような場合には，ユーザ・インターフェースなどがわかりにくく，インストールしたものの，その先がわからないといったことになりがちかと思われるので，Virtual PCを使用した方法での簡単な導入方法について解説を加えておきます．

① Virtual PCで仮想ハードディスク・イメージを作成します．単独アプリケーションとしてのVirtual PCを使用することで，まず仮想化されたOSを構築してみると，その動作について理解しやすいと思います．そこで，はじめにVirtual PCをインストールします．その後，Virtual PCで，新規仮想PCを作成し，このときに仮想ハードディスクを作成します．次に仮想PCを起動し，ホストPC上のCD-ROMなどを仮想PCに割り当てて，CD-ROMなどからゲストOSをインストールすることで仮想PCのイメージができあがります．

② 次に，「Virtual Server Administration Website」を起動し，新規の仮想PCを構築します．手順はVirtual PCとほぼ同じです．Virtual Serverにて仮想PCを構築したら，①で作成したイメージ・ファイル(拡張子VHD)をVirtual Serverで新規作成したファイルに名前を変更して上書きしてみましょう．このとき，ファイルを上書きするのではなく，仮想PCの追加でVirtual PCの仮想PCを使用することも可能です．

③ 正常にVirtual Serverの仮想ハードディスク・ファイルに移行できていれば，「Virtual Server Administration Website」から，仮想PCを起動することができるようになっています．

● 起動している仮想PCを制御する

起動している仮想PCを実際に操作するには「Virtual Machine Remote Control Client」，もしくは「VMRCコンポーネント」を使用します．ブラウザ上で仮想PCを操作するActiveXコンポーネントも付属していますが，Virtual PCの操作に近い，アプリケーションとしての「Virtual Machine Remote Control Client」でも良いでしょう．

「Virtual Machine Remote Control Client」を使用するには，「Virtual Server Administration Website」のスクリーン・サムネイルをダブルクリックして「VMRC Server」を使用可能にします．ここの設定でクライアントとの通信を行います．サーバとして常時使用し，クライアント・コントロールが必要でないのであれば，セキュリティ確保のためにVMRCを使用しない設定にしておくとよいでしょう．

● 今後のサーバ仮想化について

ハードウェアをより効率よく運用しようとする際，仮想化技術はその一つの選択肢であると筆者は考えています．また，Intel 925Xチップセットのように，ハードウェアがより一新されてくることで，旧OSの新システムへの移行にかかわるリスクを減らし，平行運用をよりよくする手段ではないかとも考えます．Virtual Serverでは，COMインターフェースにより各種の外部プログラム制御なども公開されているので，単純に他OSを動作させるVirtual PCとは一線を画した仕上がりになっていると思います．

皆さんもこれを機会にサーバ仮想化をしてみませんか?

参考URL
(1) マイクロソフト社 Virtual Serverページ
http://www.microsoft.com/japan/windowsserversystem/virtualserver/

ひろはた・ゆきお OpenLab.

シニアエンジニアの技術草子 四拾弐之段

◆非理法権天

旭 征佑

● サラリーマンと兵法

「非理法権天」――ひりほうけんてん．太公望秘伝の書である「六韜」のうち，有名な「虎の巻」に登場する格言だ．遣唐使によって初めて日本に伝えられたといわれ，江戸時代以降広く民衆に広まったとされる．今でも年輩の人なら知らない人はいないほど有名なことばらしい．その一方で若い人にとっては，はて，あまり聞いたこともない言葉かもしれない．

非理法権天は，「非は理に克(か)たず，理は法に克たず，法は権に克たず，権は天に克たず」と読む．簡単に説明すると次のようなことだ．我々はみずからの判断基準で誤ったこと――「非」か，正しいこと――「理」か判断する．しかし，規則や法律――「法」があると，たとえ悪法でも法のほうが優先される．

しかし，そんな法でも，権力者――「権」は，曲げてしまうことができる．いわゆる無法だ．しかし，世の中は何もかも天道(てんどう)とも呼ばれる天地自然の法則――「天」に支配されており，これが最後に克つというのだ．つまり，真に正しいことを見極め，それを信じて行動するのが克つ秘訣という．

「克つ」というのは「勝つ」という意味より広く，克己（おのれに克つ）ことから始まり，果ては経営・政治にいたる数々の難題を克服する，ということをも意味する兵法の最終目標だ．マネージメントが要求される中堅サラリーマンに，兵法書の人気が高いのもこういった理由からであろう．非理法権天は，その兵法のもっとも真髄ともいえることばなのだ．この辺は，中国を中心に普及した東洋的な思想で，西洋にはない．西洋では良くも悪くも自己中心的な個人主義が一般的で，このような思想は受け入れらないに違いない．

● 過去の戦いと封印

非理法権天で有名なのは，なんといってもこれを旗印に戦ったともいわれる稀代の知将，楠木正成(くすのきまさしげ)だ．楠木正成は，鎌倉時代末期わずか数百の兵で，現在の大阪府金剛山に千早城を築いて立てこもり，数万ともいわれる幕府軍の攻勢になんと数年もの間，一歩も引けを取らなかったという．これを見た新田義貞や足利尊氏はついに幕府を見限って造反，鎌倉幕府は滅びる．しかし，後に足利尊氏が室町幕府を開き，数万の軍勢で攻め寄せてくると，楠木正成は京を守るべく覚悟の出陣，わずか700騎で果敢に抗戦し，最後には自害して果てている．

太平記では，足利尊氏が敗将である楠木正成の戦いをたたえて首級(しゅきゅう)（武将の首のこと）を子に送ったが，これに腹を立てた弟の足利直義が，楠木正成の墓を掘り返したという記述がある．しかし中から出てきたのは，非理法権天と書いた紙切れだった．直義は「さすが正成，死してまでも墓堀りを見抜いたか」と大いに悔しがったと書かれている．

正成の千早城で策謀の限りを尽くした戦いや，最後の戦いでの正成親子の「桜井の別れ」の名シーンは，以前は小学校の教科書にも必ず登場していたらしい．これが，年輩の人の間では有名な理由である．

そんな名将が，若い人たちの間でまったく知れ渡っていないのは，痛ましい過去の戦争との関わりで封印されてしまったことが原因だろう．天皇を最後まで守って華々しい最後を遂げた正成の思想は，戦前の日本には極めて好都合だった．それだけではない，非理法権天の天は，天皇という意味に解された．さらに，楠木正成の家紋は菊水といって，上半分は今の天皇家の家紋ほとんどそのもの，下半分は流水を現していた．

軍人の出征のはなむけでは，非理法権天のたれ幕が華々しく掲げられた．また片道だけの燃料を積んで最後の出撃をする戦艦「大和」の艦上にも掲げられたのも非理法権天だった．家紋の菊水という名称も，特攻隊の隊名や，沖縄周辺の特攻攻撃の秘密作戦名称に使われていた．

室町幕府の時代，幕府の敵だった楠木正成の評価は公式にはあまりよくない．しかし太平記など当時の民衆が残した記録によって楠木正成は極めて高い評価をされている．室町幕府崩壊後は，公式に多くの研究がなされ，寛政の改革で有名な松平定信も評価しなおして書に残したし，あの水戸黄門は碑まで建てさせたという．

そんな楠木正成が，戦争の思想制御に利用されたというだけで，封印されてしまったのは，まことに惜しい．

読者も見ただろうか．皇居外苑の騎馬上に勇姿をたたえ，圧倒的な存在感でそびえ立っている巨大な銅像は，なにを隠そう楠木正成である．

● 海賊版との戦い

マイクロソフトは2000年ごろから世界的に海賊版との戦いを開始したという記憶がある．しかし，実際にマイクロソフトを

シニアエンジニアの技術草子



本気にさせたのは，2001年11月に，Windows XPの発売開始前に海賊版が出回ってしまうという前代未聞の不祥事が起きてからではないかと思う．2002年4月に中国マイクロソフトの総裁が，「マイクロソフトの最大の敵は海賊版で，しかも最大の被害を受けているのは中国マイクロソフトだ」（中国知識産権報）と語った．そして，その4か月後には，日本でもネット・オークションなどで海賊版を販売していたユーザの逮捕，告訴劇が次々と始まり，現在も続いている．

そんなマイクロソフトの悩みの一つが，海賊版へのセキュリティ・パッチであろう．Computer Times報として，マイクロソフトは「海賊版ユーザにも安全性を確保する必要がある」と，SP2の提供に踏み切ったと報道されていた（NIKKEI IT BUSINESS & NEWS，5月9日）．

しかし，これは米マイクロソフトの担当者によって否定されたようだ．Windows & .NET Magagine Networkによれば，「今まで報道されていることは誤りだ．SP2はインストール時に，OSのプロダクトIDが海賊版に使われている既知のIDと一致するかどうかを調べる．もしも一致すれば，SP2はインストールされない」（Nikkei BP ITpro，5月12日）という．これは，今後もセキュリティ・パッチが海賊版に供給されない可能性を意味する．事実は確認する手段がないのでわからないのだが，最後の報道がさらに否定されたというニュースは聞かない．

海賊版は，P2Pネットワークやオークションで世界各国に数えきれないほど流れている．中には，企業などで自分が海賊版を使用していることすらわからない人も大勢いるだろう．「マイクロソフト」，「海賊版」で検索すると，Yahoo!で約14,000件，Googleで約6,300件もあった．もう，海賊版は文化かもしれない．海賊版を支持するわけではないが，こんなに数のある海賊版が，ウィルスに感染してインターネットに撒き散らしたり，攻撃の踏み台のターゲットにされたりすると，世界中のユーザに大きな迷惑がかかるのは誰の目にも明らかだ．

マイクロソフトの古川亨氏（現マイクロソフト執行役員，最高技術責任者）とは何度か話をしたことがある．彼の人の話をよく聞く謙虚さ，そして行動力は，今でも強く記憶に残っている．非理法権天は東洋から発した思想だ，西洋の会社マイクロソフトには簡単に理解できないに違いない．その点，マイクロソフトに強い影響力をもつ古川氏や日本マイクロソフトの役割は大きいのかもしれない．マイクロソフトの判断が最終的に翻えされることを期待したい．



＊　　　＊

話は変わるが，あるメジャーなアンチ・ウィルス・ソフトをオンラインで購入した．「今月末までの特別キャンペーン，オンラインなら同額で1年半のライセンス!」という半自動的に何度も出てくるたいへんお得な案内に，ついに根負けして購入してしまった．しかし，数日後にがっかりすることになる．「最後の6日間だけ，オンラインなら同額で2年のライセンス!」．話が違うではないか．今度は「最後の1日だけ，3年のライセンス!」なんてのが来るかもしれない，などと冗談混じりで同僚と話しているが，実はけっこうショックが大きい．

この会社ももちろん西洋の会社だ．ぜひ非理法権天ということばを教えてあげたいなどと思うのは，筆者の修行が足りないせいなのかもしれない．

あさひ・しょうすけ　テクニカル・ライタ
イラスト　森 祐子

Engineering Life in

出会いには不向きのシリコンバレー

読者や日本の知り合いからいただく質問に,「シリコンバレーのエンジニアはそんなに忙しいのに，異性との出会いがあるんですか?」とか,「恋人や結婚相手はどうやって探すんですか?」というものがある．シリコンバレーは外から見ると未来都市か何かのようなイメージがあるが，実際はサンフランシスコ郊外の田舎街が発達したところだ．データだけを見ると大都市に分類されるかもしれないが，日本で想像されるものとのギャップも大きいだろう．そこで今回は，そんなエンジニア達のプライベート・ライフに少し踏み込んでみたい．

☆ 国勢調査や統計データから見たシリコンバレー

シリコンバレーはエンジニアの街であるのは確かで，どのハイテク企業を見ても男性が多いのは実感できる．

2000年に行われた全米国勢調査によると，アメリカの人口は2億8,140万人になったそうだ．男女の比率は，女性が少しだけ多く，男性:女性= 96.3:100となっている．この比率が男性に傾いていた都市が20あるのだが，その中にシリコンバレーの中心部，サニーベール市が含まれていた.「やっぱり男だらけのシリコンバレー!?」を実証するようなデータで，男性:女性=106:100になっている．もっとも，老人から子供まで含んだ数字なので，どれぐらいインパクトがあるかはわからない．

しかし，その一方でサンノゼを中心としたシリコンバレーは，子持ちの家族が多い都市でもある．アメリカでは，ニューヨークやシカゴなどの大都市になると結婚をしていないシングルや母子家庭が増える傾向がある．そんな中でシリコンバレーは大都市にしては子持ちが多い街だ．筆者が考えるに，これは海外から移民してきた住民の結婚率と出産率が一般アメリカ人より高いからではないかと思う．

☆ シリコンバレーの女性は何を求めているのか?

では，一方の女性はどう見ているのであろうか?　シングルの女性からするとシリコンバレーやサンフランシスコ・ベイエリアは厄介なところだそうだ．たしかに男性は多いが，彼女達からするとやっぱりエンジニアとか技術系の会社に勤めている男性はとっつき難いらしい．アメリカでいうGeek(図1)，日本語で言うとオタクっぽい男性が多いのが懸念されるらしい．たしかに高給取りで仕事も安定しているけれど…ステレオ・タイプなイメージの，ダサいとか不潔だとか，また趣味も偏っていてゲームが大好きだとか日本のアニメにやけに詳しいとかいうシリコンバレーのエンジニアはたしかに多数いる．しかし，そうでないエンジニアも多い．ハリウッド俳優並みにハンサムで，ジムでしっかりマッチョな身体を作り，ヨーロッパ製高級車で仕事に通い，オペラやクラシック音楽が趣味とかいう人もいる．しかし，仕事がらアグレッシブになりすぎて自分の自慢話しかできない，などということもある．一見ステレオ・タイプには当てはまらないが，対人関係や人付き合い下手になっていたケースだ．彼女達からすると，男性からたくさん声がかかるが，なかなか良い人が見つからないというのが多くの意見だ．

では，彼女達は何を求めているのか?　これは難しい質問だ．日本での男女関係と比べると，アメリカでは女性もしっかり仕事を持って社会的に進出しているので，女性も存在感があり，どうしても女性が強いという感覚がある．つまり，結婚しても亭主関白な男性はほとんど見つからないということだし，結婚する前の交際時期から女性に尽くさなければならない．また，生活コストが高いシリコンバレーでは相当稼ぎが良くないといけないと感じるのかもしれない．さらに，人種[注1]や信じている宗教[注2]が絡むと相手探しがよりいっそう難しくなる．

ちなみにシリコンバレーには女性のGeekも存在する．男性エンジニアのように子供のころから数学や理科が得意だったという人達だ．パソコンを自作したり，やけにプログラミングに詳しいとかいうところは男性エンジニアと同じである．しかし，必ずしも相手が男性Geekでなくてもよいらしく，男性の好みは一般的な女性と大差ないようだ．

☆ 出会いの場所とシーンは?

シリコンバレーで働くシングルの男女はどうやって出会うのか?　いろいろ考えられる中からリストアップしてみた．

1) 学校からの知り合い
2) 職場や仕事
3) 知り合いや兄弟，親戚からの紹介
4) バーやクラブなどで
5) 教会や礼拝をする場所で
6) スポーツ・ジム，趣味のサークル，ボランティア活動を通じて
7) 結婚紹介所などの仲介会社を通じて
8) インターネット(チャット・ルーム，メル友，出会い系サイトなど)

多くは日本とあまり変わらないかと思うが，中にはアメリカ独特の出会いの場がある．まずは，1番目だが，ここには日米の格差はないと思う．学校を出たての若手エンジニアであれば，学校からの知り合いなどのネットワークで出会いがよくあると思う．その次は，職場や仕事を通じてだ．これも日米格差なくあると思う．ただし，シリコンバレーでは職場の出会いがこじれて事件になることがある．サンノゼ市は大都市であるがドラッグやらギャングなどの問題が非常に少ない街である．これは20年前ぐらい

注1：海外から来たエンジニア，インド人や中国人は，自分と同じ人種・国の人でないと困るとか…白人だけれど，アジア系の女性と結婚したいなど，いろいろ細かくあるようだ．

注2：日本ではあまりなじみがないが，同じ宗教や礼拝場に行くことを必須とするカップルが多い．

から地元の警察が徹底してこういう問題に対応した結果である．しかし，シリコンバレーでの数少ない殺人事件の原因は，ギャングやドラッグとは関係のない，知人間での殺人やドメスティック・バイオレンス，家族間による暴力がエスカレートしたケースだ．ニュース性が高いものとしては，10年ぐらい前にあった電子部品系の会社でのストーカー事件があった．エンジニアが職場で知り合った女性に交際を求めたが，断られてしまった．このエンジニアは優秀な技術者ではあったが，対人関係があまり得意でない，ステレオ・タイプなGeekであった．その後ストーカー行為をして会社で問題になった．そして，このエンジニアはほかの理由で会社を去ったが，数週間後，仕返しに会社へ拳銃を持って乗り込んで銃の乱射事件を起こし，上司や関係者が数人死亡した．女性は撃たれたが大丈夫だった．本人は地元警察のSWATに射殺された．これは大事件になった例だが，シリコンバレーにも多少しつこく付け回したりする男性がいるので，女性のほうもガードが固い．ちなみに6番目のスポーツ・ジムは，このコラムでも少し書いたことがあるが，社交の場となっている．ここでも同じようにしつこく女性を付け回す輩が多く，筆者も通っているジムで見かけたことが何度かある．

4番目のバーとかクラブであるが，これはアメリカらしいものではないだろうか？　ハリウッド映画に出てくるワンシーンのようなもので，独身の男女が新しい出会いを求めて仕事帰りや土日におしゃれをしてクラブに出かけるわけだ．一方，5番目の教会や礼拝する場所を通じてというケースでは，「良い人は飲み屋に行きません…ちゃんと教会に行くのです」というアメリカのお婆ちゃんの知恵からきている．実際に教会では独身者向けのイベントなどを多く用意している．出会ってくれて，結婚して，そのまま家族で礼拝を続けてくれれば教会側としても嬉しいということなのだろう．6番目のボランティア活動も社交の場としては人気がある．たとえば，同じ環境保護関係のボランティア活動では趣味や政治的[注3]な傾向が同じなので，ボランティア活動をして人の役に立ち，また恋人探しもできるので，アメリカでは一石三鳥とも考えられている．

7番目と8番目は合計で10億ドル近くになるビジネスに成長しているといわれる．7番の結婚紹介所はオーソドックスな紹介，つまりお見合いまではいかないが，相手を紹介するところまでセッティングするようなビジネスだ．ただ，最近では形が変わり，パーティや野外でのサイクリングを多数で行うタイプが多くなっており，紹介所もコンサルタントを置いて細かく服装や身だしなみ，しゃべり方のコーチングをしてくれるところも出ているようだ．最近話題になっているのは3分デートというイベントで，同じ数の男女が参加するタイプだ．必ず全員に話をする機会が設けられるという特典があるが，3分間に限られている．全員と話ができたら解散し，その後は各自連絡を取り合うというシステムらしい．気の短いアメリカ人にとっては人気のようだ．



図1　ステレオ・タイプなGeekの例

8番の中のインターネットの出会い系サイトは3億ドル相当のビジネスに成長しているといわれており，さまざまなサービスが展開されている．アメリカらしい点は，自分の性格や好みを細かく入力するようになっていることだ．日本の携帯電話系のサイトと違い，未成年者がトラブルにあったとか犯罪につながったケースがそれほどないので，アメリカでは社会的に受け入れられていると感じる．

☆ 結論：やっぱり難しい場所?!

シリコンバレーのように世界各国からエンジニアや優秀な人材が集まるとやはり好みも千差万別になり，一筋縄でまとめることがなかなか難しい．生い立ちや教育，そして母国語，人種や宗教ベースの差，政治の好み，日本では考えられないようなパラメータが複雑に入り混じる．それでいてエンジニアリングや技術が三度の飯よりも好きな男性の多い街であるということは出会いもなかなか難しい．

トニー・チン　htchin@attglobal.net　WinHawk Consulting

注3：政治も相手を選ぶ基準となるのがアメリカだ．民主党か共和党かそれ以外なのか，支持してる党の右よりか左よりか，中絶を支持するか，銃器に関して法規制をするべきか，移民を増やすべきか制限すべきか…など，アメリカ人の世論を二分する議論が多く，恋人や夫婦間でもこれを同じ傾向にしたいと考える人が多い．

NEWPRODUCTS

●マルチ・プラットホーム FPGA

Virtex-4 シリーズ

- 特定アプリケーションの個別の要求に合わせて，ロジック，メモリ，パラレルおよびシリアル I/O，組み込みプロセッサ，高性能 DSP，クロック機能，ハード IP，ミックスド・シグナル，その他の機能ブロックなどのコア機能の異なった組み合わせを提供．
- 「Virtex-4 LX」は，一般的なロジック主体のアプリケーション用に構成されており，組み込みのブロック RAM，ディジタル・クロック機能ブロック，XtremeDSP/数値演算機能を装備．
- 「Virtex-4 SX」は，無線通信，ビデオ，マルチメディアおよびオーディオなどの高性能な信号処理アプリケーション用に構成されており，高速なリアルタイム信号処理ができるように設計されている．
- 「Virtex-4 FX」は，高速シリアル・インターフェースおよび組み込みプロセッサを含む，ネットワーキング，ストレージ，通信および組み込みアプリケーション向けに最適化されたデバイス．600Mbps～11.1Gbps までの任意の転送速度をサポートするマルチギガビット・シリアル・トランシーバをサポート．

●価格：下記へ問い合わせ

■ザイリンクス（株）
TEL：03-5321-7740 FAX：03-5321-7762

●32 ビット CISC マイコン

H8SX/1582F

- 5V の単一電源で動作．
- 最大動作周波数は 48MHz で，48MIPS の高処理性能を実現．
- 10 ビット A-D 変換器を 2 ユニットで 16 チャネル，16 ビットのタイマ・パルス・ユニットを 2 ユニットで 12 チャネル搭載．
- 各ユニットは，ユーザの使用目的に応じて，独立して動作させることが可能．
- 高速同期シリアル通信が可能な，シンクロナス・シリアル・コミュニケーション・ユニットを 3 チャネル搭載しており，各チャネルは独立して同期シリアル通信を行うことが可能．

●サンプル価格：￥1,680



■（株）ルネサス テクノロジ
TEL：03-5201-5276

●1 チップ・システム LSI

MB91403, MB91402

- IPv6 に対応した 10Mbps および 100Mbps の Ethernet 機能を搭載した FR コアの 1 チップ・システム LSI．
- ネットワーク上のパケットから，必要なパケットだけを通過させる機能を持ったハードウェアを搭載しているため，CPU に負荷をかけることなく通信ができる．
- 256K バイトの ROM と 64K バイトの RAM を内蔵することにより，外付けメモリの個数削減が可能となり，システムのコスト・ダウンや基板サイズの縮小に貢献．
- チップ内でデータ処理を行えるため，セキュリティ対策にも有効．
- 暗号処理の専用ハードウェア回路として搭載している DES・3DES に加え，次世代標準暗号化方式の AES を搭載し，セキュリティを強化．
- 認証機能ハードウェア回路を内蔵．
- MB91402 はセキュリティ機能を外したモデル．

●サンプル価格：MB91403 ￥2,000
MB91402 ￥1,500

■富士通（株）
TEL：042-532-1397
E-mail：edevice@fujitsu.com

●第 4 世代 Bluetooth チップ

BlueCore4

- 外部フラッシュ・メモリ用「BlueCore4-External」およびマスク ROM 用「BlueCore4-ROM」の 2 タイプを用意．
- 既存の Bluetooth v1.1/v1.2 仕様デバイスと完全な下位互換性を持ち，PSK（Phase Shift Keying）を採用することで，高速なデータ転送速度を実現．
- 最大 2.1Mbps のデータ転送速度を実現．
- パケット・ペイロードのシンボルが，無線リンクに送信される場合，各シンボルあたりより多くのビット転送が可能となる．
- 48K バイトのオンチップ RAM をサポートし，性能強化された EDR（Enhanced Data Rate）の処理用追加バッファとして使用される．

●価格：下記へ問い合わせ



■シーエスアール（株）
TEL：03-5328-1400 FAX：03-5328-1403

●USB2.0 カード・リーダ・コントローラ

USB2224, USB2223

- USB ホスト・コントローラとフラッシュ・メディア・カード間のインターフェースを提供する，フラッシュ・メディア・カード・コントローラ．
- プログラム・メモリとして内蔵のマスク ROM もしくは，外付けフラッシュ・メモリの使用が可能．
- LED 制御などの動作について，オプション設定が可能．
- USB Bulk Mass Strage 準拠の Bootable BIOS をサポートするフラッシュ・メディアからの PC ブートが可能．
- SmartMedia，xD Picture Card，Memory Stick/PRO，Secure Digital，MultiMediaCard，CompactFlash Ⅰ&Ⅱ など多彩なメディアに対応．

●サンプル価格：$4.45（1,000 個時）



■スタンダードマイクロシステムズ（株）
TEL：03-5487-0481 FAX：03-5487-0490
URL：http://www.smsc.jp/

●クロック生成デバイス

ispCLOCK

- 七つの 5 ビット・カウンタ（入力，フィードバック，5 本の出力）を備え，出力周波数選択を微細に行える．
- ユニバーサル・ファンアウト・バッファは，バンクや周波数にかかわらず，最大 50ps のピン間スキュー仕様で，最大サイクル・ツー・サイクル出力ジッタは 100ps 以下．
- 個々のクロック・ネット出力スキュは，基板上のクロック・ネットワーク・トレース長の差異を補うために，200ps の遅延感覚で制御可能．
- リファレンス入力とユニバーサル・ファンアウト・バッファは広く普及しているシングル・エンドおよび差動ロジック標準をそれぞれの電源電圧でサポート．

●価格：$18.25（1,000 個時）



■ラティスセミコンダクター（株）
TEL：03-3342-0701 FAX：03-3372-0750

●1チップ・オーディオ・プロセッサ

STV82x7ファミリ

- ヨーロッパおよびアジアでの地上波テレビ放送のためのアナログ・オーディオ伝送信号の自動検出とデモジュレーションに必要なリソースのすべてを提供.
- イコライゼーション，ラウドネス，ビーバー，ボリューム，バランス，サウンド効果などのオーディオ・プロセッシング効果に加えて，バーチャル(2.1)とリアル(5.1)のマルチ・チャネル性能をサポート.
- 個別のディジタル・オーディオ処理ブロックが，ヘッドホンとラウド・スピーカ出力用に提供.
- フルレンジのTVオーディオ構成に対応.
- 2スピーカ・システムでは，Virtual Dolby SurroundとVirtual Dolby Digitalが使用でき，フルマルチ・チャネルでは，Dolby Pro Logic IとPro Logic IIを使用できる.
- SRS WOW，SRS TruSurround XT，ST OmniSurroundなどのサウンド強化やバーチャル効果をサポート.

●サンプル価格：¥8,000

■STマイクロエレクトロニクス
TEL：03-5783-8340 FAX：03-5783-8216

●解像度変換LSI

IP00C716

- ドット・マトリクス型の表示デバイスに必要不可欠な，カラー・ディジタル画像の拡大，縮小を1チップで実行.
- UXGA-HDTVまでの広範囲な画像入力，UXGA画像出力に対応.
- 7セット分の12ビットγ補正テーブルを搭載し，YUV入力信号に対して，輝度の10ビット処理を実現することで，滑らかな階調表現が可能.
- 外付けのフレーム・メモリ機能により，フレーム・レート変換やフレーム同期などの機能を実現可能.
- 縦横独立な倍率設定ができ，4:3画像から16:9画像への変換が実現可能.

●価格：下記へ問い合わせ



■アイチップス・テクノロジー（株）
TEL：06-6492-7277 FAX：06-6492-7388

●PLLシンセサイザ

ADF4007, ADF4154

-「ADF4007」周波数シンセサイザは，固定周波数(7.5GHz)のインテジャNタイプのPLLシンセサイザで，RF側は7.5GHz，PFD側は120MHzまでの動作が可能．低ノイズのディジタルPFD，高精度のチャージ・ポンプ回路および分周器/プリスケーラで構成．分周器/プリスケーラの値は，2個の外部制御ピンにより，四つの値(8/16/32/64)のうちの一つに設定可能．基礎周波数の分周器は常に2に設定されており，外部基準入力周波数は最高240MHzまで可能.
-「ADF4154」は，500MHzから4.0GHzのフラクショナルNタイプのPLL周波数シンセサイザで，2.7～3.3Vの電圧範囲で動作．オプションのVpを使用することで，調節電圧の拡大が可能．プログラマブル・チャージ・ポンプ電流，二つのプログラマブル・プリスケーラ係数4/5および8/9，さらにディジタル・ロック検出およびパワーダウン・モードなどの特徴を持つ.

●価格：ADF4007 $2.78(10,000個時)
ADF4154 $2.93(10,000個時)

■アナログ・デバイセズ（株）
TEL：03-5402-8125

●RFID用メモリ

LRI64

- 64ビットのOTPユーザ・メモリと読み取り専用UID番号を備えており，13.56MHz RFID製品のISO 15693標準に準拠.
- 8ビット×16ブロック構成で，最初の8ブロックには64ビットの読み取り専用UIDを含む.
- 内部同調コンデンサとKILLコマンドを含む.
- 衝突防止シーケンスの各操作を可能にするAFIレジスタをサポート.
- ユーザが使用可能なWORM領域として，56ビット領域を提供.
- 128ビットという総メモリ・サイズにより，96ビットEPCのオンデマンドのデータ・コーディングのサポートと登録を行うことが可能.

●サンプル価格：¥11(10,000,000個時)



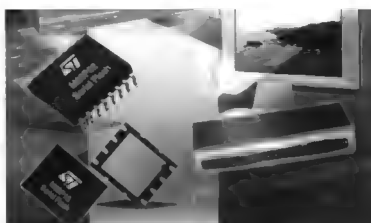
■STマイクロエレクトロニクス（株）
TEL：03-5783-8240 FAX：03-5783-8216

●シリアル・フラッシュ・メモリ

M25P16, M25P32

-「M25P16」は16Mビット，「M25P32」は32Mビットの CMOSプロセスを使用して構築された，シリアル・フラッシュ・メモリ.
- 最大50MHzのバス周波数での動作が可能.
- 1Mビットのコードを，21msでダウンロードすることが可能.
-「M25P16」は512Kバイト・セクタの32セクタ，「M25P32」は512Kバイト・セクタの64セクタとして構成.
- SPIバス互換シリアル・インターフェースとして構成された，アドレス，データ，制御用の4ピンのみを使用.

●サンプル価格：M25P16 ¥320(1,000個時)
M25P32 ¥500(1,000個時)



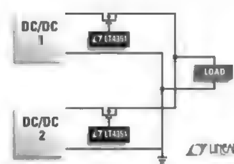
■STマイクロエレクトロニクス（株）
TEL：03-5783-8240 FAX：03-5783-8216

●MOSFETダイオードORコントローラ

LT4351

- 複数電源アプリケーションにおいて，OR接続ダイオードに代わる低損失デバイス.
- 外付けNチャネルMOSFETにより，高電流が可能.
- MOSFETゲート・ドライブ用の昇圧レギュレータ電源を内蔵.
- 1.2～18Vの広い入力範囲をサポート.
- 高速スイッチングによる，MOSFETゲート制御.
- 低電圧入力および過電圧入力を検出.
- STATUSおよびFAULT#出力により，モニタが可能.
- MOSFETゲート・クランプを搭載.
- 電源が供給されるとMOSFETがオンになるため，電源から負荷までの電圧降下を最小限に抑えることが可能.

●サンプル価格：¥270(1,000個時)



■リニアテクノロジー（株）
TEL：03-5226-7291 FAX：03-5226-0268

■弊誌では新製品に関するニュース・リリースを募集しております.
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュース・リリース係
FAX：(03)5395-2127，E-mail：mngnews@cqpub.co.jp （編集部）

NEWPRODUCTS

●ディジタル・アイソレータ

IL261, IL610/610A, IL611/611A

- 「IL261」は磁気を用いた5チャネル・ディジタル信号アイソレータ．最高110Mbd，遅延時間10ns，パルス幅歪み2ns(typ.)，チャネル間スキュ2ns(typ.)の性能を有する．送信4チャネル，受信1チャネルで構成されており，A-Dコンバータにはジッタがなく，エッジ・タイミングの正確なクロックを供給．
- 「IL610」および「IL611」は，受動入力型アイソレータ.「IL611」はデュアル・チャネルの単一方向デバイスで，反転動作または非反転動作に設定することが可能．いずれも標準のCMOS出力をサポート．
- 「IL610A」および「IL611A」は，オープン・ドレイン出力となっているため，ワイヤードOR構成が可能．
- 動作温度範囲は－40～＋85℃，データ転送速度は最大20MBaud，コイル・ドライブ電流の代表値は7mA，3.3Vまたは5Vの出力電流などの特徴を持つ．

●価格：下記へ問い合わせ

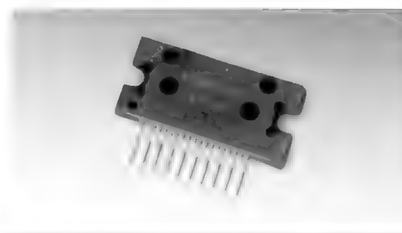
■（株）ロッキー
TEL：03-3228-4511 FAX：03-3388-1391

●ハイブリッドIC

STK453-030

- 独自のIMST(絶縁金属基板技術)とトランスファ・モールド・パッケージ・プロセスを組み合わせ，新規開発の電圧増幅段ICによる30W×2チャネルのパワーアンプ用ハイブリッドIC．
- パワーアンプに必要なほとんどの保護機能を内蔵．
- 小型パッケージ(29.2×12.9×4.5mm)により，セットの小型化が可能．
- 新開発の半導体デバイスの構成により，ハイレベルなサウンドを実現．
- 従来品と比較して，外付け部品を約30%削減．
- Stand-By回路の内蔵により，電源投入時のポップ・ノイズを低減．

●サンプル価格：¥200



■三洋電機（株）
TEL：03-3837-6265 FAX：03-3837-6378

●リアルタイム・クロックモジュール

S4E16728

- 長波帯標準電波を受信し，自動的に時刻同期を行うためのアンテナを含めたすべての機能を内蔵しており，シリアル・インターフェース方式のリアルタイム・クロックとして機能．
- JJY標準時刻電波(40kHz，60kHzの2局に対応)に対して感度調整済み．
- 一般的なクロック同期式と同じシリアル・インターフェースからデータを取り込むだけで，自動時刻調整機能付きリアルタイム・クロックを構成可能．
- 従来の自動時刻修正機能付きリアルタイム・クロック・モジュールと比較して小型であり，アプリケーションへの組み込みが容易．
- 低消費電流設計により，乾電池駆動のアプリケーションの構成に適する．
- 強制受信入力端子，受信状態モニタ端子により，アプリケーション組み込み後の設置場所での受信状態確認が行える．

●サンプル価格：¥5,000

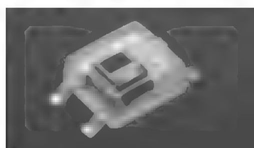
■セイコーエプソン（株）
TEL：042-587-1286

●照度センサ

NaPiCa

- シリコン・フォト・ダイオードを使用した，視感度に近い感度特性を持った光学フィルタと，光電流を増幅させるための電流アンプを内蔵し，1チップに集積した照度センサ．
- ピーク感度波長は，580nm．
- 蛍光灯100lxの明るさの場合，平均260μAの電流を出力．
- カドミウムを使用していないため，環境問題に厳しいヨーロッパ市場でも使用可能．
- 動作電圧は1.5～6V DCで，バッテリ駆動に適する．
- 逆電圧は－0.5～8Vで，光電流は5mA．
- 許容損失は，40mW．
- 動作温度は－30～85℃，保存温度は－40～100℃．

●価格：オープン



■松下電工（株）
TEL：06-6908-1131
URL：http://www.nais-j.com/

●小型ネットワーク・モジュール

Ansel Module 開発キット

- 富士通製ネットワークLSI「MB91401」，ROM，RAM，Ethernetポート，シリアルポート，エクスターミナルI/Fを搭載した開発キット．
- プロトコル・スタック「KASAGO IPv6」と「KASAGO IPSec」を搭載しているので，信頼性の高い，高速セキュア通信が可能．
- 既存システムのIPv4/IPv6ネットワーク機能拡張が容易に可能．
- ハードウェアに最適化したTCP/IPプロトコル・スタックや各種ドライバをセットで提供．
- 提供されるプロトコル・スタックは，IPv4/IPv6デュアルスタックなので，IPv6へのスムーズな移行が可能．
- IPSecの暗号，認証処理をハードウェア暗号エンジンが処理することで，処理を高速化．

●価格：下記へ問い合わせ

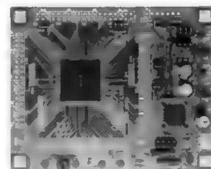
■（株）エルミックシステム
TEL：045-664-5171 FAX：045-650-1021
E-mail：info@elmic.co.jp

●ブレッド・ボード

CSP-027シリーズ

- アルテラ社のFPGAであるACEX1Kを用いたブレッド・ボードの完成品．
- 電源回路，リセット回路，クロック，ISP可能コンフィグレーションROMを実装．
- 4層基板にグレードアップしながらも，シンプルな構成を実現．
- コンフィグレーションROMは，アルテラ社のEPC2を標準装備しており，ByteBlasterMVなどで書き換えが可能．
- 全ピンを外部に引き出せるようになっている．
- 汎用LEDを2個使用しており，ジャンパで切り離しが可能．

●価格：下記へ問い合わせ



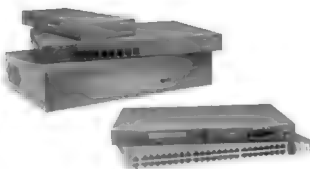
■（有）ヒューマンデータ
TEL：072-620-2002 FAX：072-620-2003
E-mail：sales2@hdl.co.jp

●遠隔監視ツール

SecureLinx SLK, SecureLinx SLC

- 「SecureLinx SLK」は，UNIX，Linux，Windowsサーバの KVM（キーボード，ビデオ，マウス）信号をネットワーク上へ変換し，IP経由で遠隔地からサーバの操作を実現するメンテナンス・ツール．完全なハードウェア・ソリューションのため，サーバのBIOSレベルまで直接操作が可能．サーバ，クライアントともにソフトウェアをインストールする必要がなく，汎用的なブラウザからの操作が可能．
- 「SecureLinx SLC」は，サーバやIT機器のコンソール・ポートへIP経由でアクセスし，機器の遠隔操作を実現するメンテナンス・ツール．

●価格：下記へ問い合わせ



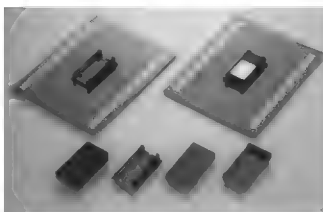
■日本ラントロニクス（株）
TEL：03-3780-7025 FAX：03-3780-7026

●SSOP用ソケット

SSC02シリーズ

- 同社のSOP用ソケットで実績のある，2点接触コンタクト挟み方式を採用したSSOP用の70ピン・ソケット．
- IC挿入時，エッジ接触によるワイピング効果で，ICテールの酸化皮膜や汚れを除去．
- ガスタイト接続により，高い接触信頼性を有する．
- コンタクトでICリードを保持するため，耐振動性，耐衝撃性に優れている．
- ICを挿入する際には，専用冶具を押すだけなので，高い作業性を有する．
- IC抜去時には，同社SOP用ソケット向け抜去冶具を兼用して使用可能．

●サンプル価格：¥160



■ケル（株）
TEL：042-374-5801 FAX：042-374-5887
E-mail：kikaku@kel.co.jp

●ビデオ・サーバ

AXIS 241Q AXIS 241S

- 「AXIS 241Q」は，ビデオ入力端子を4個装備しており，最大4台までのアナログ・カメラを接続でき，Ethernet経由で4チャネルから同時にフル・フレーム・レートのディジタル・ビデオ・データを送信することが可能．
- 「AXIS 241S」は，アナログ・ビデオ端子1個とEthernet接続端子に加えて，アナログ・ビデオ出力端子を1個装備しており，Y/C-BNC変換ケーブルを利用して分離したY/Cビデオ信号の受信が可能．
- 独自開発の画像圧縮チップ「ARTPEC-2」を搭載．

●価格：¥176,400（AXIS 241Q）
¥102,900（AXIS 241S）



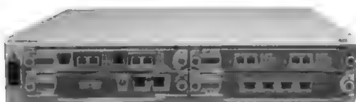
■アクシスコミュニケーションズ（株）
TEL：03-6716-7850 FAX：03-6716-7851
E-mail：info@axiscom.co.jp
URL：http://www.axiscom.co.jp/

●ネットワーク向けテスト・ソリューション

N2Xソリューション

- 「OmniBER XM」，「ルータテスタ900」，「SANテスタ」の三つの機能を搭載することで，SONET/SDHのアラーム試験，IPパフォーマンスおよびプロトコル試験，SANパフォーマンス試験に1台で対応可能．
- IP試験とSONET/SDH試験を行う場合，同じコントローラで双方の制御および試験を同時に行うことができ，統合された試験が可能．
- SONET/SDH試験用カード，IPトラフィックおよびパフォーマンス試験用カード，SAN試験用カードなど5種類のインターフェースカードの中から必要なカードを装備することで，さまざまな試験に対応できる構成となっている．

●価格：¥2,600,000
（小型スイッチ，ルータ試験用構成）
¥6,600,000（SONET/SDH試験用構成）



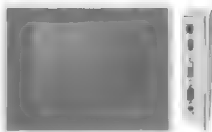
■アジレント・テクノロジー（株）
TEL：0120-421-345

●フラットパネル・ディスプレイ

IPC-DT/S61VT-DC1, IPC-DT/S65VT-DC1

- 「IPC-DT/S61VT-DC1」はパネルマウント可能なアルミ化粧フェイス付きタイプ，「IPC-DT/S65VT-DC1」はアルミ化粧なしの組み込みタイプ．
- 広視野角，高輝度タイプの262,144色表示可能な液晶を装備．
- 汎用のアナログRGB入力対応．
- タッチパネルは，RS-232-CまたはUSB接続が可能．
- USBタッチパネルI/F使用時は，1ホストに最大8台までのマルチ・タッチパネル構成が可能．
- 入力された画面サイズを，液晶のドット構成に合わせて表示する，オート・スケーリング機能を搭載．

●価格：IPC-DT/S61VT-DC1 ¥113,400
IPC-DT/S65VT-DC1 ¥108,150



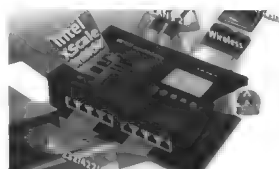
■（株）コンテック
TEL：03-5628-9286 FAX：03-5628-9344
E-mail：tsc@contec.co.jp

●通信プラットホーム

UC-7400シリーズ

- RISCベースの通信プラットホーム．
- 八つのRS-232/422/485シリアル・ポート，デュアル10/100MbpsのEthernetポート，PCMCIA，コンパクト・フラッシュ・インターフェースおよび拡張無線通信機能をサポート．
- Linuxがプリインストールされているため，プログラム開発が容易．
- ソフトウェア・コードを修正することなく，GNUクロス・コンパイラを使用することで，デスクトップPC用のソフトウェアを容易に移植することが可能．
- インテルXscale IXP-422 266MHzプロセッサを搭載．

●価格：下記へ問い合わせ



■山下システムズ（株）
TEL：03-5700-2121 FAX：03-5700-0076
E-mail：info@misnet.co.jp

■弊誌では新製品に関するニュース・リリースを募集しております．
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュース・リリース係
FAX：(03)5395-2127，E-mail：mngnews@cqpub.co.jp （編集部）

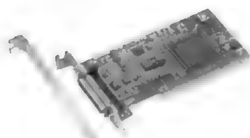
NEWPRODUCTS

●アナログ入出力ボード

ADA16-8/2(LPCI)L AD16-16(LPCI)L DA16-4(LPCI)L

- 「ADA16-8/2(LPCI)L」は，アナログ入力(16ビット/8チャネル)，アナログ出力(16ビット/2チャネル)，アナログ入出力の制御信号(6チャネル)を装備．
- 「AD16-16(LPCI)L」は，アナログ入力(16ビット/16チャネル)，アナログ入力の制御信号(3チャネル)を装備．
- 「DA16-4(LPCI)L」は，アナログ出力(16ビット/4チャネル)，アナログ出力の制御信号(3チャネル)を装備．
- いずれの機種も，ディジタル入力(4チャネル)，ディジタル出力(4チャネル)，カウンタ(32ビット/1チャネル)を搭載．

●価格：
ADA16-8/2(LPCI)L ¥51,450
AD16-16(LPCI)L ¥44,100
DA16-4(LPCI)L ¥51,450



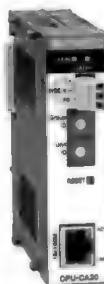
■(株)コンテック
TEL：03-5628-9286 FAX：03-5628-9344
E-mail：tsc@contec.co.jp

●I/Oコントローラ・モジュール

CPU-CA20(FIT)GY, SVR-IOA2(FIT)GY

- 「CPU-CA20(FIT)GY」は，外部信号入出力および信号処理用にRISCチップSH-4 240MHz，通信インターフェースに帯域100Mbpsの100Base-TXを搭載するコントローラ．ディジタル入出力，アナログ入出力，カウンタ入力など，接続する装置インターフェースの信号仕様に合わせて構成可能．Ethernetをベースにリモート I/O制御を実現し，点在する装置や設備を遠隔から集中制御するシステムを構築可能．
- 「SVR-IOA2(FIT)GY」は，登録した複数の「CPU-CA20(FIT)GY」を定周期で自動的に巡回，全I/O情報をマッピングする機能を搭載．マネジメント機能により，ホスト・コントローラ間のネットワークの負荷を大幅に軽減することが可能なI/Oアシスト・サーバ・ユニット．

●価格：
CPU-CA20(FIT)GY ¥37,800
SVR-IOA2(FIT)GY ¥37,800



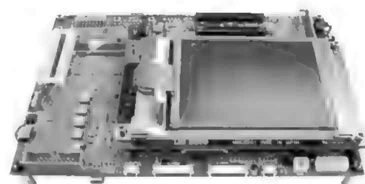
■(株)コンテック
TEL：03-5628-9286
FAX：03-5628-9344
E-mail：tsc@contec.co.jp

●端末機器開発ツール

xP-5000

- インテルPXA27xプロセッサ・ファミリ搭載．
- PDAなどの携帯端末や据え置き型情報端末，VoIP，TV電話といった製品の開発に必要な周辺デバイスを実装．
- 拡張用のバス・コネクタを用意しており，ボードのカスタマイズやOSの移植，アプリケーション開発などのシステム製品に必要なことが行える．
- ハードウェア開発ツールとして使用することで，ハードウェア/ソフトウェアの切り分けが容易になり，ハードウェアの検証期間を短縮できる．
- ハードウェア開発前にソフトウェアの開発に着手することができ，アプリケーションの開発から，製品開発期間が短縮できる．

●価格：¥344,400



■安川情報システム(株)
TEL：044-952-8913 FAX：044-952-8923

●グラフィックス開発環境

MontaVista Graphics 3.1

- 利用可能なオープンソース・テクノロジを活用し，多数のコンポーネントから単一のグラフィカル開発キットを構築．
- X Window Systemベースで，ネットワークにも対応．
- Gtk2+ツールキットを包含し，グラフィックス・アプリケーションを構築するために必要なウィジェットとツールを提供．
- フリースタイル・フォントのサポート，フォント・レンダリング用Pango，2種類のウィンドウ・マネージャ(IceWM/Matchbox)などの利用が可能．
- ARMベース，PowerPC，MIPS，SuperH，x86の各アーキテクチャをサポートし，共通のソースとバイナリ・プラットホームを提供．

●価格：下記へ問い合わせ

■モンタビスタソフトウェアジャパン(株)
TEL：03-5469-8840 FAX：03-5469-8801

●統合設計環境

Libero 6.0

- サード・パーティの設計ツールがシームレスに統合されているため，FPGAの設計効率を向上させることができる．
- Platinum，Gold，Silverの三つのエディションがあり，すべてのエディションにシンプリシティ社のシンセシス・ツール「Synplify」アクテル・エディションをバンドル．
- インターフェースからマグマ社のフィジカル・シンセシス・ツールPALACEのフル・バージョンにアクセス可能．
- GUIによって，タイミングやコンストレイントを追加できるシンプリシティのSCOPE Editionを無償でバンドル．
- 階層化されたネットリスト・ブラウザ，バッチ，コマンド・ライン・モード(Floating License用)，HDL Analystまでのインターフェースなどにアクセス可能．

●価格：下記へ問い合わせ

■アクテルジャパン(株)
TEL：03-3445-7671
E-mail：japan@actel.com

●動画圧縮・解凍ツール

PICVideo M-JPEG Codec V3

- ペガサス・イメージング社が開発した，Motion JPEGファイルをWindows Media Playerで表示するのに最適なCODECで，モーションJPEGストリームの高速圧縮・解凍機能を提供．
- 640×480ピクセル，30フレーム/sのスピードで動画をキャプチャし，カラーまたは白黒で再生する圧縮・解凍が可能．
- Adobe Premier，Windows，DirectShowを含む，ビデオ・アプリケーションと互換．
- スライダ・コントロールにより，圧縮と品質を簡単に設定可能．
- 非標準モーションJPEGストリーム用に自動調整．
- 動画再生中にコントラストや明るさの設定を変更可能．
- YV12とIYUV/I420をサポート．

●価格：¥15,750(1CPU圧縮と解凍)
¥6,300(1CPU解凍のみ)

■エクセルソフト(株)
TEL：03-5440-7875 FAX：03-5440-7876
E-mail：xlsoftkk@xlsoft.com

●画像処理ツール

Smartscan Xpress ICR/OCR/OMR V4

- 画像内に含まれる機械文字，手書き文字や光学式マークを識別し，画像内の不要部分を削除する機能を提供．
- Win32ビジュアル開発環境で，.NET，VB，Delphi，VC++，HTMLを使用したアプリケーション開発が可能．
- ALTを使用することで，最小限のフットプリントを実現し，MFCを必要としない．
- ActiveX/COMまたはVCLをホストする環境での開発が可能．
- .NETマネージド・コントロールとして，アプリケーションの配置が可能．
- 3種類の処理速度を選択可能．

●価格：
Smartscan Xpress OCR/OMR V4 ¥117,600～
Smartscan Xpress ICR/OCR/OMR V4 ¥220,500～

■エクセルソフト（株）
TEL：03-5440-7875 FAX：03-5440-7876
E-mail：xlsoftkk@xlsoft.com

●マルチメディア・アプリケーション開発ツール

SH-MobileJ2 マルチメディア アクセラレータ・プラットホーム

- カメラやサウンド・インターフェースなどのハードウェアを制御するデバイス・ドライバ，動画と音声の同時再生やビデオ・メールなどのミドルウェア，ベースバンドLSIとのソフトウェア・インターフェースを容易にするアクセラレータAPIの3種類のソフトウェアで構成．
- ドライバ・ソフトウェアは，動作周波数133MHzの「SH-MobileJ2」を搭載したアプリケーション開発用のボード「Solution Engine」上で動作するハードウェア制御用のソフトウェア．
- アクセラレータAPIは，単体のミドルウェアをシステムとして統合したもので，約150のAPIを用意．
- 「SH-MobileJ2」に搭載しているMPEG-4アクセラレータなどのハードウェア性能を最大限に引き出せるように最適化されている．
- 多彩で高機能なマルチメディア・アプリケーションの開発が容易となるため，開発効率を向上できる．

●価格：下記へ問い合わせ

■（株）ルネサス テクノロジ
TEL：03-5201-5234

● MPEGソフト・エンコーダ

MPEG-2 Encoder6

- 独自のビット・レート割り付け技術や視覚感度変調技術をはじめとする，高圧縮化，高画質化アルゴリズムを搭載し，画質の向上を実現．
- 低ビット・レートでも，ノイズを大幅に削減して，細部まで鮮明にエンコードが可能．
- 動きの激しい画像でも，圧縮時に発生するノイズを大幅に削減．
- AV機器用規格やAPIであるDirectShow対応などによって，画像編集ソフトやDVDオーサリング・ソフトなどのアプリケーション・ソフトでの活用に適する．
- 高画質長時間録画や，画質を要求されるコンテンツ制作にも耐えうる画質での圧縮が可能．

●価格：下記へ問い合わせ

■松下電器産業（株）
TEL：06-6900-9724 FAX：06-6900-9654
E-mail：psc@ml.jp.panasonic.com

●オーディオCODEC

MIPS Consumer Audio Platform

- MIPS32命令セットに最適化されており，Dolby Digital，MPEGオーディオ，Windows Media AudioやSRS TruSurround XT技術に必要なミドルウェアを提供．
- SoC設計において，ホストCPUもしくは専用のMIPS RISCコプロセッサ上でオーディオ・アルゴリズムを動作させるための柔軟性を実現．
- オーディオ・ストリームの仕切りを排除し，デュアルRISC/DSPソフトウェア・ツール・チェーンやプロセッサ・アーキテクチャをサポートすることが可能になる．
- コアのシンセサイザブル性を利用することで，消費電力，ダイ・サイズ，もしくは市場要求に依存した性能などさまざまなオプションを独自のSoCで構成することを可能にする．

●価格：下記へ問い合わせ

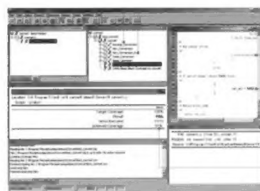
■ミップス・テクノロジーズ
TEL：03-5733-9544 FAX：03-5733-9545

●単体テスト・ツール

Cantata++

- ウィザード機能により，テストケースを基にしたテスト用コード・テンプレートを自動生成．
- ラッピング技術により，関数，ハードウェアなどの外部オブジェクトのシミュレーションが可能．
- テスト・ハーネスによる，テスト結果の検証，出力が可能．
- カバレッジ自動測定機能を搭載．
- 静的解析による，ソフトウェアの品質測定を実現．
- テスト結果，テスト・カバレッジ結果をレポートするグラフィカル・インターフェース「Cantata++ Studio」機能を追加．

●価格：下記へ問い合わせ



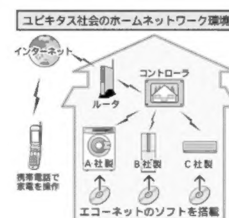
■（株）エーアイコーポレーション
TEL：03-3493-7981 FAX：03-3493-7993
E-mail：sales@aicp.co.jp

●通信ミドルウェア

SYSTEM COMPONENT for ECHONET

- 白物家電，住設機器を相互接続する次世代ホーム・ネットワーク規格である，エコーネットに対応した通信ミドルウェア・ソフト．
- 家電製品本体の開発コストの低減が可能．
- センサ，リモコンなどメモリ容量の少ない機器に搭載が可能で，8ビットCPUで動作する，ハードウェア・スペックを最小限に抑えたネットワーク家電の製品化を可能にする．
- 各種マイコンへの移植に対応．
- 約14KバイトのROM，および約0.4KバイトのRAMで動作する．
- ハードウェアとして，100msがカウントできるタイマ/カウンタが別途必要．

●価格：オープン価格



■安川情報システム（株）
TEL：044-952-8919 FAX：044-952-8923
E-mail：echonet@ysknet.co.jp

■弊誌では新製品に関するニュース・リリースを募集しております．
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュース・リリース係
FAX：(03)5395-2127，E-mail：mngnews@cqpub.co.jp （編集部）



INFORMATION
海外・国内イベント/セミナー情報



海外イベント

8/2-5 **LinuxWorld Conference & Expo**
Moscone Center, San Francisco, CA, USA
IDG
http://www.linuxworldexpo.com/

8/16-19 **COMDEX Korea 2004**
COEX Atlantic Hall & Conference Center, Seoul, Korea
COMDEX Korea
http://comdex.chosun.com/

9/7-11 **ITU TELECOM ASIA 2004**
Busan Exhibition & Convention Center, Busan, Korea
ITU TELECOM
http://www.itu.int/ASIA2004/

9/21-23 **Wescon**
Anaheim Convention Center, Anaheim, CA, USA
IEEE
http://www.wescon.com/

9/26-10/1 **MobiCom 2004**
Loews Philadelphia Hotel, Philadelphia, PA, USA
ACM SIGMOBILE
http://www.sigmobile.org/mobicom/2004/

10/4-8 **PCB Design Conferences East**
Expo Center of New Hampshire, Manchester, NH, USA
UP Media Group
http://www.pcbeast.com/

10/13-16 **electronicAsia 2004**
Hong Kong Convention & Exhibition Centre, Hong Kong, China
Hong Kong Trade Development Council
http://www.electronicasia.com/

国内イベント

9/15-17 第6回 自動認識総合展
東京ビッグサイト(東京都江東区有明)
(社)日本自動認識システム協会
http://www.autoid-expo.com/

9/22-25 A&Vフェスタ 2004
パシフィコ横浜(神奈川県横浜市西区)
(社)日本オーディオ協会
http://www.jas-audio.or.jp/

9/24-26 東京ゲームショウ 2004
幕張メッセ(千葉県千葉市美浜区)
(社)コンピュータエンターテイメント協会
http://tgs.cesa.or.jp/

10/5-9 CEATEC JAPAN
幕張メッセ(千葉県千葉市美浜区)
日本エレクトロニクスショー協会
http://home.jesa.or.jp/jp/exhibitions/ceatec/

10/13-15 第7回 関西 機械要素技術展/設計製造ソリューション展
インテックス大阪(大阪府大阪市住之江区)
リード・エグジビション・ジャパン(株)
http://www.mtech-kansai.jp/
http://www.dms-kansai.jp/

10/20-22 FPD International 2004
パシフィコ横浜(神奈川県横浜市西区)
日経BP
http://expo.nikkeibp.co.jp/fpd/ja/index.html

11/10-12 第15回 マイクロマシン展
科学技術館(東京都千代田区)
(財)マイクロマシンセンター
http://www.micromachine.jp/

開催日，イベント名，開催地，問い合わせ先の順

日程はすべて予定です．問い合わせ先にご確認のうえ，お出かけください．

セミナー情報

Linux ネットワークプログラミング
開催日時 : 8月6日(金)
開催場所 : エイチアイ研修室(東京都目黒区東山)
受講料 : 49,000円(税込:テキスト代含む)
問い合わせ先:(株)エイチアイ ICP事業部，☎03)3719-8155，FAX(03)5773-8661
http://icp.hicorp.co.jp/seminar/linux/linux_network.asp

失敗しないための!! UML導入の留意点と効果の出し方
開催日時 : 8月9日(月)
開催場所 : SRCセミナールーム(東京都新宿区高田馬場)
受講料 : 50,400円(税込)
問い合わせ先:(株)ソフト・リサーチ・センター，☎03)5272-6071，FAX(03)5272-6345
http://www.src-j.com/seminar_no/24/24_144.htm

Cプログラマのための C++ 徹底入門
開催日時 : 8月9日(月)〜10日(火)
開催場所 : エイチアイ研修室(東京都目黒区東山)
受講料 : 92,000円(税込:テキスト代含む)
問い合わせ先:(株)エイチアイ ICP事業部，☎03)3719-8155，FAX(03)5773-8661
http://icp.hicorp.co.jp/seminar/c-vc/vc_c++.asp

これからセキュリティに携わる方のための
わかりやすい!! セキュリティ国際標準ISO15408とISO17799入門解説
開催日時 : 8月19日(木)〜8月20日(金)
開催場所 : SRCセミナールーム(東京都新宿区高田馬場)
受講料 : 79,800円(税込)
問い合わせ先:(株)ソフト・リサーチ・センター，☎03)5272-6071，FAX(03)5272-6345
http://www.src-j.com/seminar_no/24/24_146.htm

ソフト開発における要求の仕様化と管理法
開催日時 : 8月24日(火)
開催場所 : オームビル(東京都千代田区)
受講料 : 55,125円(税込:1口で1社3名まで受講可)
問い合わせ先:(株)トリケップス，☎03)3294-2547，FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/sem/c040824n.htm

表計算ソフトによるディジタル信号処理技術
開催日時 : 8月24日(火)〜26日(木)
開催場所 : 高度ポリテクセンター(千葉県千葉市美浜区)
受講料 : 30,000円(税込)
問い合わせ先: 高度ポリテクセンター事業課，☎043)296-2582，FAX(043)296-2585
http://www.apc.ehdo.go.jp/

JPEG2000 符号化方式の基礎
開催日時 : 8月25日(水)
開催場所 : オームビル(東京都千代田区)
受講料 : 55,125円(税込:1口で1社3名まで受講可)
問い合わせ先:(株)トリケップス，☎03)3294-2547，FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/sem/c040825n.htm

C/C++ 言語で学ぶ遺伝的アルゴリズムプログラミング
開催日時 : 8月25日(水)〜27日(金)
開催場所 : 高度ポリテクセンター(千葉県千葉市美浜区)
受講料 : 30,000円(税込)
問い合わせ先: 高度ポリテクセンター事業課，☎043)296-2582，FAX(043)296-2585
http://www.apc.ehdo.go.jp/

Linux GUI プログラミング
開催日時 : 8月30日(月)
開催場所 : エイチアイ研修室(東京都目黒区東山)
受講料 : 46,000円(税込:テキスト代含む)
問い合わせ先:(株)エイチアイ ICP事業部，☎03)3719-8155，FAX(03)5773-8661
http://icp.hicorp.co.jp/seminar/linux/clinuxgui.asp

デジタル画像のフォーマットとカラー処理・変換技術
開催日時 : 8月30日(月)〜31日(火)
開催場所 : オームビル(東京都千代田区)
受講料 : 65,625円(税込:1口で1社3名まで受講可)
問い合わせ先:(株)トリケップス，☎03)3294-2547，FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/sem/c040830n.htm

画像処理/産業応用画像処理技術
開催日時 : 9月8日(水)〜10日(金)
開催場所 : 高度ポリテクセンター(千葉県千葉市美浜区)
受講料 : 30,000円(税込)
問い合わせ先: 高度ポリテクセンター事業課，☎043)296-2582，FAX(043)296-2585
http://www.apc.ehdo.go.jp/seminar/

USB2.0 ターゲット・システムの設計事例
開催日時 : 9月9日(木)
開催場所 : CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料 : 13,000円(税込)
問い合わせ先: エレクトロニクス・セミナー事務局，☎03)5395-2125，FAX(03)5395-1255
http://it.cqpub.co.jp/eSeminar/

DSP 活用技術
開催日時 : 9月15日(水)〜17日(金)
開催場所 : 高度ポリテクセンター(千葉県千葉市美浜区)
受講料 : 30,000円(税込)
問い合わせ先: 高度ポリテクセンター事業課，☎043)296-2582，FAX(043)296-2585
http://www.apc.ehdo.go.jp/seminar/

ディジタル信号処理入門
開催日時 : 9月17日(金)
開催場所 : CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料 : 13,000円(税込)
問い合わせ先: エレクトロニクス・セミナー事務局，☎03)5395-2125，FAX(03)5395-1255
http://it.cqpub.co.jp/eSeminar/

DSP によるディジタル・フィルタ入門
開催日時 : 9月18日(土)
開催場所 : CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料 : 13,000円(税込)
問い合わせ先: エレクトロニクス・セミナー事務局，☎03)5395-2125，FAX(03)5395-1255
http://it.cqpub.co.jp/eSeminar/

読者の広場

Interfaceへの声



2004年7月号特集「MIPSプロセッサ徹底活用研究」に関して

▷ルネサステクノロジのSH-3＆SH-4のユーザですが，組み込み用64ビット・マイコンを手がけようと考えています．モトローラのPowerPCだと消費電力が大きいし，やはりMIPSかなと思っています．NECのV_R5500シリーズはなかなか強力そうで楽しみです．やはりパイプラインはスーパスカラ方式がベストでしょう．

（白石 隆）

アンケートの結果

興味のあった記事（2004年7月号で実施）

①プロローグ なぜMIPSなのか
②第1章 MIPSアーキテクチャの変遷と現状
③第2章 MIPSアーキテクチャの詳細
④Appendix1 フル・シンセサイザブル・コアMIPS32 24Kの概要
⑤Appendix2 MIPSアーキテクチャ・エミュレータ simips
⑥第5章 V_Rシリーズの概要とV_R5701の詳細
⑦MMUなしでも動作するLinux――はじめて使うμClinux（第1回）
⑧Linux用PCカード・デバイス・ドライバの作成
⑨移り気な情報工学
⑩第6章 TXシリーズとT-Engine/TX4956の概要
⑪フリーソフトウェア徹底活用講座（第16回）
⑫やり直しのための信号数学（第25回）
⑬第3章 PMC-Sierra RMシリーズの概要とRM7900＆RM9000x2GLの詳細
⑭第4章 Alchemyソリューション SoCの詳細
⑮プログラミングの要（第13回）
⑯開発技術者のためのアセンブラ入門（第26回）
⑰Appendix3 クイックロジック QuickMIPSの概要
⑱Appendix4 IDT RC32434統合コミュニケーション・プロセッサ

特集『MIPSプロセッサ徹底活用研究』についてのアンケートの結果

Q1 MIPS系プロセッサを使ったCPUボードなどを設計されたことがありますか？
①はい（33%） ②いいえ（77%）

（Q1で「はい」と回答された方に質問です）
Q2 CPUの動作周波数はどの程度ですか？
①100MHz未満（50%） ②～200MHz（0%）
③～400MHz（50%） ④～800MHz（0%）
⑤800MHz超（0%）

Q3 プロセッサに関連した内容で，どんな記事を希望しますか？（複数回答可）
①アーキテクチャ解説（24%）
②システム・ボード設計方法（24%）
③周辺コントローラの使い方（19%）
④アセンブラの使い方（5%）
⑤IPL/BIOS/ファームウェアの作成方法（5%）
⑥OS移植方法（10%）
⑦ソフトウェア開発環境構築（14%）
⑧その他（0%）

特集担当デスクから

☆今月は，ディジタル信号処理技術の基礎的な部分に焦点を当てた，数学的な根拠に重点を置いた特集でしたが，いかがでしたか？ ご意見，ご感想，ご要望などを読者アンケートはがきか，本誌のWebページ（http://www.cqpub.co.jp/interface/）に設置されている掲示板でお伝えいただけると幸いです．
☆本誌では，過去に何度もこの分野を特集しています．古いものでは，今から約20年も前の1985年4月号にて「はじめてのディジタル信号処理」と題して，DSPの使用例なども織り交ぜながら（なんと，今回の特集第1章で登場したμPD7720とTMS320C10を取り上げている！），今回の特集と同様，「移動平均」やら「z変換」，「ディジタル・フィルタ」などの基礎的な部分も多く解説してあります．これらの基礎的な内容は，当然ながら，今も85年当時もまったく変わりありません．つまり，今，こういった技術の基礎となる部分を身に付けておけば，20年先も役立つはずです．

Interface

次号予告

2004年10月号は
8月25日発売です

USBホスト&ターゲット・システム設計技法

USBターゲット/USBホスト/OHCI/UHCI/On-The-Go/PictBridge

USBは現在のPCであれば必ず備えているインターフェースと考えても良いほど，標準インターフェースとして定着した．また，PC周辺機器として一般的なものは，ありとあらゆるものがUSBで接続可能になっている．

そして現在では，これまでPCの周辺機器として接続してきたディジタル・カメラやプリンタが，PCを介さずとも接続可能になり，写真を直接プリント・アウトできるようになっている．さらに，セット・トップ・ボックスやホーム・サーバのような情報家電機器などにも，USBホスト機能が搭載されることで，機能や使い勝手がますます向上していくだろう．

次号の特集では，各社のUSBターゲット・デバイス，組み込み機器向けUSBホスト・コントローラ内蔵マイコン，USB On-The-Go対応デバイスを取り上げ，実際のサンプル・プログラムを示しながら詳しく解説する．

★次号には，『コネクタ・ピン配置 大全集』が別冊付録として付きます！

編集後記

●日本車がアメ車に勝っていたのは燃費の良さだけではなく，仕上げの丁寧さ，狂いのない鋳造，ガタのこないドア，塗装の丁寧さといった品質面だった．重要なのは苦情のでる割合をつねに低く抑える地道な努力をして，ユーザの信頼を勝ちえたことだった．而して今すべての日本車とは言えなくなってしまった．（檀）

●SPAMと架空請求に悩まされている毎日．最近では，.ruとか.yuなどからもSPAMが来るようになっている．どういう具合にメアドが流れているのだろうか？　また，架空請求は，直電でも来るようになった．携帯からかけてきているので，捨て番号だろう．対応するにも時間がかかって面倒だ．困ったものだ．（=IO）

●菓子を食べないと集中して仕事ができない．そんなわけで，今週はものすごい量の甘い菓子を食べまくった．肥満対策は「飲むだけで食べたもののカロリーを80%もカットしてくれる」という魔法のようなサプリメント（高かった）もあわせて摂取．しかし効果はなかった．たった1週間で3kgも肥えた．金返せっ!!（もみ）

●初めて梅酒をつくってみた．梅と氷砂糖とホワイト・リカー（焼酎とどう違うの？）をビンに入れるだけで数か月後にはおいしい梅酒ができる予定．しかし，製造元である私は最近急に酒に弱くなった．ワイン・グラス1杯でとろけてしまう．先日は酔っ払ってしまい，友人相手にずっと敬語でしゃべっていた．（太陽熱）

●自宅の無線LANアクセス・ポイントが修理から戻ってきました．修理伝票を見たら「不具合再現せず」とか書いてますよ!?　でも保障期間中だったので「本体&ACアダプタを交換します」だそうで…とりあえず繋ぎ直したら問題なく家庭内LANが復旧しましたが…よくよく見ると，ACアダプタが前のより大きくなってるゾ…（M）

●（先月の続き）筋トレを続けて1か月，見た目にはそれほど変化がないものの，最初に借りた筋トレ・グッズが柔らかく感じるということは，効果が出てきたということかな？　そうなると，大胸筋だけでなく背筋や腹筋も鍛えてみたいと欲も出てくるわけで…．まあ，深入りせずにできる範囲で鍛えていきますか．（み）

●蒸し暑く寝苦しい夜が続く季節になってきましたが，ビールが美味しい季節でもありますよね．ビア・ガーデンとかに行って，開放的な雰囲気の中で枝豆と一緒に冷たいビールを飲みたいと思っています．でも，毎年そう思って実現できていないので多分今年も行けないかも…．とにかく夏のビールは最高です（笑）．（Y2）

●都内の某公園には，革ジャン+リーゼント姿で踊る「ローラー族」が集まる一角があります．以前からたまり場になっていたのですが，最近，人数が増えたみたいです．アンプも新品になって音量もアップ．ひそかにリバイバル・ブームなのでしょうか…夏の暑さにも負けない彼らの気合いは評価したいと思います．（と）

お知らせ

■ **読者の広場**

本誌に関するご意見・ご希望などを，綴じ込みのハガキでお寄せください．読者の広場への掲載分には粗品を進呈いたします．なお，掲載に際しては表現の一部を変更させていただくことがありますので，あらかじめご了承ください．

■ **投稿歓迎**

本誌に投稿をご希望の方は，連絡先（自宅/勤務先）を明記のうえ，テーマ，内容の概要をレポート用紙1～2枚にまとめて「Interface投稿係」までご送付ください．メールでお送りいただいても結構です（送り先はsupportinter@cqpub.co.jpまで）．追って採否をお知らせいたします．なお，採用分には小社規定の原稿料をお支払いいたします．

■ **本誌掲載記事についてのご注意**

本誌掲載記事には著作権があり，示されている技術には工業所有権が確立されている場合があります．したがって，個人で利用される場合以外は，所有者の許諾が必要です．また，掲載された回路，技術，プログラムなどを利用して生じたトラブルについては，小社ならびに著作権者は責任を負いかねますので，ご了承ください．

本誌掲載記事をCQ出版（株）の承諾なしに，書籍，雑誌，Webといった媒体の形態を問わず，転載，複写することを禁じます．

■ **コピー・サービスのご案内**

本誌バックナンバの掲載記事については，在庫（原則として24か月分）のないものに限りコピー・サービスを行っています．コピー体裁は雑誌見開きの，複写機による白黒コピーです．なお，コピーの発送には多少時間がかかる場合があります．

- コピー料金（税込み）
  1ページにつき100円
- 発送手数料（判型に関わらず）
  1～10ページ：100円，11～30ページ：200円，31～50ページ：300円，51～100ページ：400円，101ページ以上：600円
- 送付金額の算出方法
  総ページ数×100円+発送手数料
- 入金方法
  現金書留か郵便小為替による郵送
- 明記事項
  雑誌名，年月号，記事タイトル，開始ページ，総ページ数
- 宛て先
  〒170-8461　東京都豊島区巣鴨1-14-2
  CQ出版株式会社　コピー・サービス係
  （TEL：03-5395-4211，FAX：03-5395-1642）

■ **お問い合わせ先のご案内**

- 在庫，バックナンバ，年間購読送付先変更に関して
  販売部：03-5395-2141
- 広告に関して
  広告部：03-5395-2133
- 雑誌本文に関して
  編集部：03-5395-2122

記事内容に関するご質問は，返信用封筒を同封して編集部宛てに郵送してくださるようお願いいたします．筆者に回送してお答えいたします．

Interface

©CQ出版（株）　2004　振替　00100-7-10665
2004年9月号　第30巻　第9号　通巻第327号）
2004年9月1日発行（毎月1日発行）
定価は裏表紙に表示してあります

発行人／増田久喜
編集人／山形孝雄
編集／大野典宏　村上真紀　山口光樹　大竹友美　小林由美子
デザイン・DTP／クニメディア株式会社
表紙デザイン／株式会社プランニング・ロケッツ
本文イラスト／森　祐子　神崎真理子
広告／澤辺　彰　中元正夫　菅原利江

発行所／CQ出版株式会社　〒170-8461　東京都豊島区巣鴨1-14-2
電話／編集部（03）5395-2122　FAX／（03）5395-2127
広告部（03）5395-2133　URL http://www.cqpub.co.jp/interface/
販売部（03）5395-2141　E-mail supportinter@cqpub.co.jp
CQ Publishing Co.,Ltd.／1-14-2 Sugamo, Toshima-ku, Tokyo 170-8461, Japan
印刷／クニメディア株式会社　美和印刷株式会社
製本／星野製本株式会社


日本ABC協会加盟誌
（新聞雑誌部数公査機構）　ISSN0387-9569

本書に記載されている社名，および製品名は，一般に開発メーカの登録商標または商標です．なお本文中では ™，®，©の各表示を明記しておりません．

Printed in Japan

本誌は有名書店ほか，コンビニエンス・ストア「セブン-イレブン」でも予約購読ができます．詳しくは，お店にお問い合わせください．